Nearly-Linear Time Algorithms for Preconditioning and Solving Symmetric, Diagonally Dominant Linear Systems*

Daniel A. Spielman
Department of Computer Science
Program in Applied Mathematics
Yale University

Shang-Hua Teng
Department of Computer Science
Boston University

September 17, 2009

Abstract

We present a randomized algorithm that, on input a symmetric, weakly diagonally dominant n-by-n matrix A with m non-zero entries and an n-vector $\boldsymbol{b}$, produces an $\tilde{\boldsymbol{x}}$ such that $\left\|\tilde{\boldsymbol{x}} - A^{\dagger}\boldsymbol{b}\right\|_A \leq \epsilon \left\|A^{\dagger}\boldsymbol{b}\right\|_A$ in expected time

$$m \log^{O(1)} n \log(1/\epsilon).$$

The algorithm applies subgraph preconditioners in a recursive fashion. These preconditioners improve upon the subgraph preconditioners first introduced by Vaidya (1990). For any symmetric, weakly diagonally-dominant matrix A with non-positive off-diagonal entries and $k \geq 1$, we construct in time $m \log^{O(1)} n$ a preconditioner of A with at most

$$2(n-1) + (m/k) \log^{O(1)} n$$

non-zero off-diagonal entries such that the finite generalized condition number $\kappa_f(A, B)$ is at most k. If the non-zero structure of the matrix is planar, then the condition number is at most

$$O\left((n/k) \log n \log\log^2 n\right),$$

and the corresponding linear system solver runs in expected time

$$O(n \log^2 n + n \log n \ (\log\log n)^2 \log(1/\epsilon)).$$

Similar bounds are obtained on the running time of algorithms computing approximate Fiedler vectors.

*This paper is the last in a sequence of three papers expanding on material that appeared first under the title "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems" [ST04]. The second paper, "Spectral Sparsification of Graphs" [ST08c] contains algorithms for constructing sparsifiers of graphs, which we use in this paper to build preconditioners. The first paper, "A Local Clustering Algorithm for Massive Graphs and its Application to Nearly-Linear Time Graph Partitioning" [ST08b] contains graph partitioning algorithms that are used to construct sparsifiers in the second paper.

This material is based upon work supported by the National Science Foundation under Grant Nos. 0325630, 0324914, 0634957, 0635102 and 0707522. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

We design an algorithm with nearly optimal asymptotic complexity for solving linear systems in symmetric, weakly diagonally dominant (SDD_0) matrices. The algorithm applies a classical iterative solver, such as the Preconditioned Conjugate Gradient or the Preconditioned Chebyshev Method, with a novel preconditioner that we construct and analyze using techniques from graph theory. Linear systems in these preconditioners may be reduced to systems of smaller size in linear time by use of a direct method. The smaller linear systems are solved recursively. The resulting algorithm solves linear systems in SDD_0 matrices in time almost linear in their number of non-zero entries. Our analysis does not make any assumptions about the non-zero structure of the matrix, and thus may be applied to the solution of the systems in SDD_0 matrices that arise in any application, such as the solution of elliptic partial differential equations by the finite element method [Str86, BHV04], the solution of maximum flow problems by interior point algorithms [FG04, DS08], or the solution of learning problems on graphs [BMN04, ZBL+03, ZGL03].

Graph theory drives the construction of our preconditioners. Our algorithm is best understood by first examining its behavior on Laplacian matrices—the symmetric matrices with non-positive off-diagonals and zero row sums. Each n-by-n Laplacian matrix A may be associated with a weighted graph, in which the weight of the edge between distinct vertices i and j is $-A_{i,j}$ (see Figure 1). We precondition the Laplacian matrix A of a graph G by the Laplacian matrix B of a subgraph H of G that resembles a spanning tree of G plus a few edges. The subgraph H is called an *ultra-sparsifier* of G, and its corresponding Laplacian matrix is a very good preconditioner for A: The finite generalized condition number $\kappa_f(A, B)$ is $\log^{O(1)} n$. Moreover, it is easy to solve linear equations in B. As the graph H resembles a tree plus a few edges, we may use partial Cholesky factorization to eliminate most of the rows and columns of B while incurring only a linear amount fill. We then solve the reduced system recursively.

$$\begin{bmatrix} 1.5 & -1.5 & 0 & 0 \\ -1.5 & 4 & -2 & -0.5 \\ 0 & -2 & 3 & -1 \\ 0 & -0.5 & -1 & 1.5 \end{bmatrix}$$

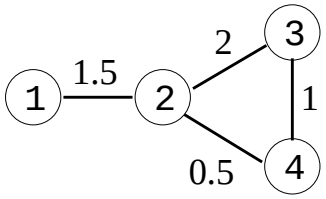

Figure 1: A Laplacian matrix and its corresponding weighted graph.

The technical meat of this paper lies in the construction of ultra-sparsifiers for Laplacian matrices, which appears in Sections 7 through 10. In the remainder of the introduction, we formally define ultra-sparsifiers, and the sparsifiers from which they are built. In Section 2, we survey the contributions upon which we build, and mention other related work. We devote Section 3 to recalling the basics of support theory, defining the finite condition number, and explaining why we may restrict out attention to Laplacian matrices.

In Section 4, we state the properties we require of partial Cholesky factorizations, and we present our first algorithms for solving equations in SDD_0-matrices. These algorithms directly solve equations in the preconditioners, rather than using a recursive approach, and take time roughly $O(m^{5/4} \log^{O(1)} n)$ for general SDD_0-matrices and $O(n^{9/8} \log^{1/2} n)$ for SDDM_0-matrices

with planar non-zero structure. To accelerate these algorithms, we apply our preconditioners in a recursive fashion. We analyze the complexity of these recursive algorithms in Section 5, obtaining our main algorithmic results. In Section 6, we observe that these linear system solvers yield efficient algorithms for computing approximate Fiedler vectors, when applied inside the inverse power method.

We do not attempt to optimize the exponent of $\log n$ *in the complexity of our algorithm.* Rather, we present the simplest analysis we can find of an algorithm of complexity $m \log^{O(1)} n \log(1/\epsilon)$. We expect that the exponent of $\log n$ can be substantially reduced through advances in the constructions of low-stretch spanning trees, sparsifiers, and ultrasparsifiers. Experimental work is required to determine whether a variation of our algorithm will be useful in practice.

1.1 Ultra-sparsifiers

To describe the quality of our preconditioners, we employ the notation $A \preccurlyeq B$ to indicate that $B - A$ is positive semi-definite. We define a SDDM_0-matrix to be a SDD_0-matrix with no positive off-diagonal entries. When positive definite, the SDDM_0-matrices are M-matrices and in particular are Stieltjes matrices.

Definition 1.1 (Ultra-Sparsifiers)**.** *A* (k, h)*-ultra-sparsifier of an* n*-by-*n SDDM_0*-matrix* A *with* $2m$ *non-zero off-diagonal entries is a* SDDM_0*-matrix* A_s *such that*

(a) $A_s \preccurlyeq A \preccurlyeq k \cdot A_s$*.*

(b) A_s *has at most* $2(n-1) + 2hm/k$ *non-zero off-diagonal entries.*

(c) The set of non-zero entries of A_s *is a subset of the set of non-zero entries of* A*.*

In Section 10, we present an expected $m \log^{O(1)} n$-time algorithm that on input a Laplacian matrix A and a $k \geq 1$ produces a (k, h)-ultra-sparsifier of A with probability at least $1 - 1/2n$, for

$$h = c_3 \log_2^{c_4} n, \tag{1}$$

where c_3 and c_4 are some absolute constants. As we will use these ultra-sparsifiers throughout the paper, we will define a *k-ultra-sparsifier* to be a (k, h)-ultra-sparsifier where h satisfies (1).

For matrices whose graphs are planar, we present a simpler construction of (k, h)-ultra-sparsifiers, with $h = O\left(\log n (\log \log n)^2\right)$. This simple constructions exploits low-stretch spanning trees [AKPW95, EEST08, ABN08], and is presented in Section 9. Our construction of ultra-sparsifiers in Section 10 builds upon the simpler construction, but requires the use of *sparsifiers*. The following definition of sparsifiers will suffice for the purposes of this paper.

Definition 1.2 (Sparsifiers)**.** *A* d*-sparsifier of* n*-by-*n SDDM_0*-matrix* A *is a* SDDM_0*-matrix* A_s *such that*

(a) $A_s \preccurlyeq A \preccurlyeq (5/4)A_s$*.*

(b) A_s *has at most* dn *non-zero off-diagonal entries.*

(c) The set of non-zero entries of A_s *is a subset of the set of non-zero entries of* A*.*

(d) For all i,

$$\sum_{j \neq i} \frac{A_s(i,j)}{A(i,j)} \leq 2 \left| \{ j : A(i,j) \neq 0 \} \right| .$$

In a companion paper [ST08c], we present a randomized algorithm `Sparsify2` that produces sparsifiers of Laplacian matrices in expected nearly-linear time. As explained in Section 3, this construction can trivially be extended to all SDDM_0-matrices.

Theorem 1.3 (Sparsification)**.** *On input an $n \times n$ Laplacian matrix A with $2m$ non-zero off-diagonal entries and a $p > 0$, `Sparsify2` runs in expected time $m \log(1/p) \log^{17} n$ and with probability at least $1 - p$ produces a $c_1 \log^{c_2}(n/p)$-sparsifier of A, for $c_2 = 30$ and some absolute constant $c_1 > 1$.*

We parameterize this theorem by the constants c_1 and c_2 as we believe that they can be substantially improved. In particular, Spielman and Srivastava [SS08] construct sparsifiers with $c_2 = 1$, but these constructions require the solution of linear equations in Laplacian matrices, and so can not be used to help speed up the algorithms in this paper. Batson, Spielman and Srivastava [BSS09] have proved that there exist sparsifiers that satisfy conditions (a) through (c) of Definition 1.2 with $c_2 = 0$.

2 Related Work

In this section, we explain how our results relate to other rigorous asymptotic analyses of algorithms for solving systems of linear equations. For the most part, we restrict our attention to algorithms that make structural assumptions about their input matrices, rather than assumptions about the origins of those matrices.

Throughout our discussion, we consider an n-by-n matrix with m non-zero entries. When m is large relative to n and the matrix is arbitrary, the fastest algorithms for solving linear equations are those based on fast matrix multiplication [CW82], which take time approximately $O(n^{2.376})$. The fastest algorithm for solving general sparse positive semi-definite linear systems is the Conjugate Gradient. Used as a direct solver, it runs in time $O(mn)$ (see [TB97, Theorem 28.3]). Of course, this algorithm can be used to solve a system in an arbitrary matrix A in a similar amount of time by first multiplying both sides by A^T. To the best of our knowledge, every faster algorithm requires additional properties of the input matrix.

2.1 Special non-zero structure

In the design and analysis of direct solvers, it is standard to represent the non-zero structure of a matrix A by an unweighted graph G_A that has an edge between vertices $i \neq j$ if and only if $A_{i,j}$ is non-zero (see [DER86]). If this graph has special structure, there may be elimination orderings that accelerate direct solvers. If A is tri-diagonal, in which case G_A is a path, then a linear system in A can be solved in time $O(n)$. Similarly, when G_A is a tree a linear system in A by be solved in time $O(n)$ (see [DER86]).

If the graph of non-zero entries G_A is planar, one can use Generalized Nested Dissection [Geo73, LRT79, GT87] to find an elimination ordering under which Cholesky factorization can be performed in time $O(n^{1.5})$ and produces factors with at most $O(n \log n)$ non-zero entries.

We will exploit these results in our algorithms for solving planar linear systems in Section 4. We recall that a planar graph on n vertices has at most $3n-6$ edges (see [Har72, Corollary 11.1 (c)]), so $m \leq 6n$.

2.2 Subgraph Preconditioners

Our work builds on a remarkable approach to solving linear systems in Laplacian matrices introduced by Vaidya [Vai90]. Vaidya demonstrated that a good preconditioner for a Laplacian matrix A can be found in the Laplacian matrix B of a subgraph of the graph corresponding to A. He then showed that one could bound the condition number of the preconditioned system by bounding the dilation and congestion of an embedding of the graph of A into the graph of B. By using preconditioners obtained by adding edges to maximum spanning trees, Vaidya developed an algorithm that finds ϵ-approximate solutions to linear systems in SDDM_0-matrices with at most d non-zero entries per row in time $O((dn)^{1.75}\log(1/\epsilon))$. When the graph corresponding to A had special structure, such as having a bounded genus or avoiding certain minors, he obtained even faster algorithms. For example, his algorithm for solving planar systems runs in time $O((dn)^{1.2}\log(1/\epsilon))$.

As Vaidya's paper was never published and his manuscript lacked many proofs, the task of formally working out his results fell to others. Much of its content appears in the thesis of his student, Anil Joshi [Jos97], and a complete exposition along with many extensions was presented by Bern *et. al.* [BGH$^+$06]. Gremban, Miller and Zagha [Gre96, GMZ95] explain parts of Vaidya's paper as well as extend Vaidya's techniques. Among other results, they find ways of constructing preconditioners by *adding* vertices to the graphs. Maggs *et. al.* [MMP$^+$05] prove that this technique may be used to construct excellent preconditioners, but it is still not clear if they can be constructed efficiently.

The machinery needed to apply Vaidya's techniques directly to matrices with positive off-diagonal elements is developed in [BCHT04]. An algebraic extension of Vaidya's techniques for bounding the condition number was presented by Boman and Hendrickson [BH03b], and later used by them [BH01] to prove that the low-stretch spanning trees constructed by Alon, Karp, Peleg, and West [AKPW95], yield preconditioners for which the preconditioned system has condition number at most $m2^{O(\sqrt{\log n \log\log n})}$. They thereby obtained a solver for symmetric diagonally dominant linear systems that produces ϵ-approximate solutions in time $m^{1.5+o(1)}\log(1/\epsilon)$. Through improvements in the construction of low-stretch spanning trees [EEST08, ABN08] and a careful analysis of the eigenvalue distribution of the preconditioned system, Spielman and Woo [SW09] show that when the Preconditioned Conjugate Gradient is applied with the best low-stretch spanning tree preconditioners, the resulting linear system solver takes time at most $O(mn^{1/3}\log^{1/2} n \log(1/\epsilon))$. The preconditioners in the present paper are formed by adding edges to these low-stretch spanning trees.

The recursive application of subgraph preconditioners was pioneered in the work of Joshi [Jos97] and Reif [Rei98]. Reif [Rei98] showed how to recursively apply Vaidya's preconditioners to solve linear systems in SDDM_0-matrices with planar non-zero structure and at most a constant number of non-zeros per row in time $O(n^{1+\beta}\log^{O(1)}(\kappa(A)/\epsilon))$, for every $\beta > 0$. While Joshi's analysis is numerically much cleaner, he only analyzes preconditioners for simple model problems. Our recursive scheme uses ideas from both these works, with some simplification. Koutis and Miller [KM07] have developed recursive algorithms that solve linear systems in SDDM_0-matrices

with planar non-zero structure in time $O(n \log(1/\epsilon))$.

2.3 Other families of matrices

Subgraph preconditioners have been used to solve systems of linear equations from a few other families.

Daitch and Spielman [DS08] have shown how to reduce the problem of solving linear equations in symmetric M_0-matrices to the problem of solving linear equations in SDDM_0-matrices, given a factorization of the M_0-matrix of width 2 [EGB05]. These matrices, with the required factorizations, arise in the solution of the generalized maximum flow problem by interior point algorithms.

Shklarski and Toledo [ST08a] introduce an extension of support graph preconditioners, called *fretsaw preconditioners*, which are well suited to preconditioning finite element matrices. Daitch and Spielman [DS07] use these preconditioners to solve linear equations in the stiffness matrices of two-dimensional truss structures in time $O(n^{5/4} \log n \log(1/\epsilon))$.

For linear equations that arise when solving elliptic partial differential equations, other techniques supply fast algorithms. For example, Multigrid methods may be proved correct when applied to the solution of some of these linear systems [BHM01], and Hierarchical Matrices run in nearly-linear time when the discretization is nice [BH03a]. Boman, Hendrickson, and Vavasis [BHV04] have shown that the problem of solving a large class of these linear systems may be reduced to that of solving diagonally-dominant systems. Thus, our algorithms may be applied to the solution of these systems.

3 Background and Notation

By $\log x$, we mean the logarithm of x base 2, and by $\ln x$ the natural logarithm.

We define SDD_0 to be the class of symmetric, weakly diagonally dominant matrices, and SDDM_0 to be the class of SDD_0-matrices with non-positive off-diagonal entries. We define a Laplacian matrix to be a SDDM_0-matrix with with zero row-sums.

Throughout this paper, we define the A-norm by

$$\|\boldsymbol{x}\|_A = \sqrt{\boldsymbol{x}^T A \boldsymbol{x}}.$$

3.1 Preconditioners

For symmetric matrices A and B, we write

$$A \preccurlyeq B$$

if $B - A$ is positive semi-definite. We recall that if A is positive semi-definite and B is symmetric, then all eigenvalues of AB are real. For a matrix B, we let $B^\dagger$ denote the Moore-Penrose pseudo-inverse of B—that is the matrix with the same nullspace as B that acts as the inverse of B on its image. We will use the following propositions, whose proofs are elementary.

Proposition 3.1. *If A and B are positive semi-definite matrices such that for some $\alpha, \beta > 0$,*

$$\alpha A \preccurlyeq B \preccurlyeq \beta A$$

then A and B have the same nullspace.

Proposition 3.2. *If A and B are positive semi-definite matrices having the same nullspace and $\alpha > 0$, then*

$$\alpha A \preccurlyeq B$$

if and only if

$$\alpha B^{\dagger} \preccurlyeq A^{\dagger}.$$

The following proposition notes the equivalence of two notions of preconditioning. This proposition is called the "Support Lemma" in [BGH+06] and [Gre96], and is implied by Theorem 10.1 of [Axe85]. We include a proof for completeness.

Proposition 3.3. *If A and B are symmetric matrices with the same nullspace and A is positive semi-definite, then all eigenvalues of $AB^{\dagger}$ lie between λ_{min} and λ_{max} if and only if*

$$\lambda_{min} B \preccurlyeq A \preccurlyeq \lambda_{max} B.$$

Proof. We first note that $AB^{\dagger}$ has the same eigenvalues as $A^{1/2}B^{\dagger}A^{1/2}$. If for all $\boldsymbol{x} \in \text{Image}(A)$ we have

$$\lambda_{min} \boldsymbol{x}^T \boldsymbol{x} \leq \boldsymbol{x}^T A^{1/2} B^{\dagger} A^{1/2} \boldsymbol{x},$$

then by setting $\boldsymbol{z} = A^{1/2}\boldsymbol{x}$, we find that for all $\boldsymbol{z} \in \text{Image}(A)$,

$$\lambda_{min} \boldsymbol{z}^T A^{\dagger} \boldsymbol{z} \leq \boldsymbol{z}^T B^{\dagger} \boldsymbol{z},$$

which is equivalent to $\lambda_{min} A^{\dagger} \preccurlyeq B^{\dagger}$ and

$$\lambda_{min} B \preccurlyeq A.$$

The other side is proved similarly. □

Following Bern *et. al.* [BGH+06], we define the *finite generalized condition number* $\kappa_f(A, B)$ of matrices A and B having the same nullspace to be the ratio of the largest to smallest non-zero eigenvalues $AB^{\dagger}$. Proposition 3.3 tells us that $\lambda_{min} B \preccurlyeq A \preccurlyeq \lambda_{max} B$ implies $\kappa_f(A, B) \leq \lambda_{max}/\lambda_{min}$. One can use $\kappa_f(A, B)$ to bound the number of iterations taken by the Preconditioned Conjugate Gradient algorithm to solve linear systems in A when using B as a preconditioner. Given bounds on λ_{max} and λ_{min}, one can similarly bound the complexity of the Preconditioned Chebyshev method.

3.2 Laplacians Suffice

When constructing preconditioners, we will focus our attention on the problem of preconditioning Laplacian matrices.

Bern *et. al.* [BGH+06, Lemma 2.5], observe that the problem of preconditioning SDDM_0-matrices is easily reduced to that of preconditioning Laplacian matrices. We recall the reduction for completeness.

Proposition 3.4. *Let A be a SDDM_0-matrix. Then, A can be expressed as $A = A_L + A_D$ where A_L is a Laplacian matrix and A_D is a diagonal matrix with non-negative entries. Moreover, if B_L is a Laplacian matrix such that $A_L \preccurlyeq B_L$, then $A \preccurlyeq B_L + A_D$. Similarly, if B_L is a Laplacian matrix such that $B_L \preccurlyeq A_L$, then $B_L + A_D \preccurlyeq A$.*

So, any algorithm for constructing sparsifiers or ultra-sparsifiers for Laplacian matrices can immediately be converted into an algorithm for constructing sparsifiers or ultra-sparsifiers of SDDM_0-matrices. Accordingly in Sections 9 and 10 we will restrict our attention to the problem of preconditioning Laplacian matrices.

Recall that a symmetric matrix A is *reducible* if there is a permutation matrix P for which $P^T AP$ is a block-diagonal matrix with at least two blocks. If such a permutation exists, one can find it in linear time. A matrix that is not reducible is said to be *irreducible*. The problem of solving a linear system in a reducible matrix can be reduced to the problems of solving linear systems in each of the blocks. Throughout the rest of this paper, we will restrict our attention to solving linear systems in irreducible matrices. It is well-known that a symmetric matrix is irreducible if and only if its corresponding graph of non-zero entries is connected. We use this fact in the special case of Laplacian matrices, observing that the weighted graph associated with a Laplacian matrix A has the same set of edges as G_A.

Proposition 3.5. *A Laplacian matrix is irreducible if and only if its corresponding weighted graph is connected.*

It is also well-known that the null-space of the Laplacian matrix of a connected graph is the span of the all-1's vector. Combining this fact with Proposition 3.4, one can show that the only singular irreducible SDDM_0-matrices are the Laplacian matrices.

Proposition 3.6. *A singular irreducible* SDDM_0*-matrix is a Laplacian matrix, and its nullspace is spanned by the all-1's vector.*

To the extent possible, we will describe our algorithms for solving irreducible singular and non-singular systems similarly. The one tool that we use for which this requires some thought is the Cholesky factorization. As the Cholesky factorization of a Laplacian matrix is degenerate, it is not immediately clear that one can use backwards and forwards substitutions on the Cholesky factors to solve a system in a Laplacian. To handle this technicality, we note that an irreducible Laplacian matrix A has a factorization of the form

$$A = LDL^T,$$

where L is lower-triangular and non-zero on its entire diagonal and D is a diagonal matrix with ones on each diagonal entry, excluding the bottom right-most which is a zero. This factorization may be computed by a slight modification of standard Cholesky factorization algorithms. The pseudo-inverse of A can be written

$$A^\dagger = \Pi L^{-T} D L^{-1} \Pi,$$

where Π is the projection orthogonal to the all-1's vector (see Appendix D).

When A is a Laplacian and we refer to forwards or backwards substitution on its Cholesky factors, we will mean multiplying by $DL^{-1}\Pi$ or $\Pi L^{-T} D$, respectively, and remark that these operations can be performed in time proportional to the number of non-zero entries in L.

4 Solvers

We first note that by Gremban's reduction, the problem of solving an equation of the form $A\boldsymbol{x} = \boldsymbol{b}$ for a SDD_0-matrix A can be reduced to the problem of solving a system that is twice

as large in a SDDM_0-matrix (see Appendix A). So, for the purposes of asymptotic complexity, we need only consider the problem of solving systems in SDDM_0-matrices.

To solve systems in an irreducible SDDM_0-matrix A, we will compute an ultra-sparsifier B of A, and then solve the system in A using a preconditioned iterative method. At each iteration of this method, we will need to solve a system in B. We will solve a system in B by a two-step algorithm. We will first apply Cholesky factorization repeatedly to eliminate all rows and columns with at most one or two non-zero off-diagonal entries. As we stop the Cholesky factorization before it has factored the entire matrix, we call this process a *partial Cholesky factorization*. We then apply another solver on the remaining system. In this section, we analyze the use of a direct solver. In Section 5, we obtain our fastest algorithms by solving the remaining system recursively.

The application of partial Cholesky factorization to eliminate rows and columns with at most 2 non-zero off-diagonal entries results in a factorization of B of the form

$$B = PLCL^T P^T,$$

where C has the form

$$C = \begin{pmatrix} I_{n-n_1} & 0 \\ 0 & A_1, \end{pmatrix},$$

P is a permutation matrix, L is non-singular and lower triangular of the form

$$L = \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & I_{n_1}, \end{pmatrix},$$

and every row and column of A_1 has at least 3 non-zero off-diagonal entries.

We will exploit the properties of this factorization stated in the following proposition.

Proposition 4.1 (Partial Cholesky Factorization)**.** *If B is an irreducible* SDDM_0*-matrix then,*

(a) A_1 is an irreducible SDDM_0*-matrix and is singular if and only if A is singular.*

(b) If the graph of non-zero entries of B is planar, then the graph of non-zero entries of A_1 is as well.

(c) L has at most $3n$ non-zero entries.

(d) If B has $2(n-1+j)$ non-zero off-diagonal entries, then A_1 has dimension at most $2j-2$ and has at most $2(3j-3)$ non-zero off-diagonal entries.

(e) $B^\dagger = \Pi P^{-T} L^{-T} \begin{pmatrix} I_{n-n_1} & 0 \\ 0 & {A_1}^\dagger \end{pmatrix} L^{-1} P^{-1} \Pi$, *where Π is the projection onto the span of B.*

Proof. It is routine to verify that A_1 is diagonally dominant with non-positive off-diagonal entries, and that planarity is preserved by elimination of rows and columns with 2 or 3 non-zero entries, as these correspond to vertices of degree 1 or 2 in the graph of non-zero entries. It is similarly routine to observe that these eliminations preserve irreducibility and singularity.

To bound the number of entries in L, we note that for each row and column with 1 non-zero off-diagonal entry that is eliminated, the corresponding column in L has 2 non-zero entries,

and that for each row and column with 2 non-zero off-diagonal entries that is eliminated, the corresponding column in L has 3 non-zero entries.

To bound n_1, the dimension of A_1, first observe that the elimination of a row and column with 1 or 2 non-zero off-diagonal entries decreases both the dimension by 1 and the number of non-zero entries by 2. So, A_1 will have $2(n_1 - 1 + j)$ non-zero off-diagonal entries. As each row in A_1 has at least 3 non-zero off-diagonal entries, we have

$$2(n_1 - 1 + j) \geq 3n_1,$$

which implies $n_1 \leq 2j - 2$. The bound on the number non-zero off-diagonal entries in A_1 follows immediately.

Finally, (5) may be proved by verifying that the formula given for $B^\dagger$ satisfies all the axioms of the pseudo-inverse (which we do in Appendix D). □

We name the algorithm that performs this factorization `PartialChol`, and invoke it with the syntax

$$(P, L, A_1) = \texttt{PartialChol}(B).$$

We remark that `PartialChol` can be implemented to run in linear time.

4.1 One-Level Algorithms

Before analyzing the algorithm in which we solve systems in A_1 recursively, we pause to examine the complexity of an algorithm that applies a direct solver to systems in A_1. While the results in this subsection are not necessary for the main claims of our paper, we hope they will provide intuition.

If we are willing to ignore numerical issues, we may apply the conjugate gradient algorithm to directly solve systems in A_1 in $O(n_1 m_1)$ operations [TB97, Theorem 28.3], where m_1 is the number of non-zero entries in A_1. In the following theorem, we examine the performance of the resulting algorithm.

Theorem 4.2 (General One-Level Algorithm)**.** *Let A be an irreducible n-by-n SDDM_0-matrix with $2m$ non-zero off-diagonal entries. Let B be a $\sqrt{m}$-ultra-sparsifier of A. Let $(P, L, A_1) =$ `PartialChol`(B). Consider the algorithm that solves systems in A by applying PCG with B as a preconditioner, and solves each system in B by a performing backward substitution on its partial Cholesky factor, solving the inner system in A_1 by conjugate gradient used as an exact solver, and performing forward substitution on its partial Cholesky factor. Then for every right-hand side $\boldsymbol{b}$, after*

$$O(m^{1/4} \log(1/\epsilon))$$

iterations, comprising

$$O(m^{5/4} \log^{2c_4} n \log(1/\epsilon))$$

arithmetic operations, the algorithm will output an approximate solution $\tilde{\boldsymbol{x}}$ satisfying

$$\left\| \tilde{\boldsymbol{x}} - A^\dagger \boldsymbol{b} \right\|_A \leq \epsilon \left\| A^\dagger \boldsymbol{b} \right\|_A . \tag{2}$$

Proof. As $\kappa_f(A,B) \leq \sqrt{m}$, we may apply the standard analysis of PCG [Axe85], to show that (2) will be satisfied after $O(m^{1/4}\log(1/\epsilon))$ iterations. To bound the number of operations in each iteration, note that B has at most $2(n-1)+O(\sqrt{m}\log^{c_4} n)$ non-zero off-diagonal entries. So, Proposition 4.1 implies m_1 and n_1 are both $O(\sqrt{m}\log^{c_4} n)$. Thus, the time required to solve each inner system in A_1 is at most $O(m_1 n_1) = O(m\log^{2c_4} n)$. As A is irreducible $m \geq n-1$, so this bounds the number of operations that must be performed in each iteration. □

If m is much greater than n, we could speed up this algorithm by first applying `Sparsify2` to compute a very good sparse preconditioner A_s for A, using the one-level algorithm to solve systems in A_s, and then applying this solver to A by iterative refinement.

When the graph of non-zero entries of A is planar, we may precondition using the the algorithm `UltraSimple`, presented in Section 9, instead of `UltraSparsify`. As the matrix A_1 produced by applying partial Cholesky factorization to the output of `UltraSimple` is also planar, we can solve the linear systems in A_1 by the generalized nested dissection algorithm of Lipton, Rose and Tarjan [LRT79]. This algorithm uses graph separators to choose a good order for Cholesky factorization. The Cholesky factorization is then computed in time $O(n_1^{3/2})$. The resulting Cholesky factors only have $O(n_1 \log n_1)$ non-zero entries, and so each linear system in A_1 may be solved in time $O(n_1 \log n_1)$, after the Cholesky factors have been computed.

Theorem 4.3 (Planar One-Level Algorithm)**.** *Let A be an n-by-n planar* SDDM_0*-matrix with m non-zero entries. Consider the algorithm that solves linear systems in A by using PCG with the preconditioner*

$$B = \texttt{UltraSimple}(A, n^{3/4}\log^{1/3} n),$$

and solves systems in B by applying `PartialChol` to factor B into $PL[I,0;0,A_1]L^TP^T$, and uses generalized nested dissection to solve systems in A_1. For every right-hand side $\boldsymbol{b}$, this algorithm computes an $\tilde{\boldsymbol{x}}$ satisfying

$$\left\|\tilde{\boldsymbol{x}} - A^\dagger \boldsymbol{b}\right\|_A \leq \epsilon \left\|A^\dagger \boldsymbol{b}\right\|_A \tag{3}$$

in time

$$O\left(n^{9/8}\log^{1/2} n \log(1/\epsilon)\right).$$

Proof. First, recall that the planarity of A implies $m \leq 3n$. Thus, the time taken by `UltraSimple` is dominated by the time taken by `LowStretch`, which is $O(n\log^2 n)$.

By Theorem 9.1 and Theorem 9.5, the matrix B has at most $2(n-1)+6n^{3/4}\log^{1/3} n$ non-zero off-diagonal entries and

$$\kappa_f(A,B) = O\left(n^{1/4}\log^{2/3} n \log^2\log n\right) \leq O\left(n^{1/4}\log n\right).$$

Again, standard analysis of PCG [Axe85] tells us that the algorithm will require at most

$$O\left(n^{1/8}\log^{1/2} n \log(1/\epsilon)\right)$$

iterations guarantee that (3) is satisfied.

By Proposition 4.1, the dimension of A_1, n_1, is at most $6n^{3/4}\log^{1/3} n$. Before beginning to solve the linear system, the algorithm will spend

$$O(n_1^{3/2}) = O((n^{3/4}\log^{1/3} n)^{3/2}) = O(n^{9/8}\log^{1/2} n)$$

time using generalized nested dissection [LRT79] to permute and Cholesky factor the matrix A_1. As the factors obtained will have at most $O(n_1 \log n_1) \leq O(n)$ non-zeros, each iteration of the PCG will require at most $O(n)$ steps. So, the total complexity of the application of the PCG will be

$$O\left(n \cdot \left(n^{1/8}\log^{1/2} n \log(1/\epsilon)\right)\right) = O\left(n^{9/8}\log^{1/2} n \log(1/\epsilon)\right),$$

which dominates the time required to compute the Cholesky factors and the time of the call to `UltraSimple`. □

5 The Recursive Solver

In our recursive algorithm for solving linear equations, we solve linear equations in a matrix A by computing an ultra-sparsifier B, using partial Cholesky factorization to reduce it to a matrix A_1, and then solving the system in A_1 recursively. Of course, we compute all of the necessary ultra-sparsifiers and Cholesky factorizations just once at the beginning of the algorithm.

To specify the recursive algorithm for an n-by-n matrix, we first set the parameters

$$\chi = c_3 \log^{c_4} n, \tag{4}$$

and

$$k = (14\chi + 1)^2, \tag{5}$$

where we recall that c_3 and c_4 are determined by the quality of the ultra-sparsifiers we can compute (see equation (1)), and were used to define a k-ultra-sparsifier.

We the following algorithm `BuildPreconditioners` to build the sequence of preconditioners and Cholesky factors. In Section 10, we define the routine `UltraSparsify` for weighted graphs, and thus implicitly for Laplacian matrices. For general irreducible SDDM_0-matrices A, we express A as a sum of matrices A_L and A_D as explained in Proposition 3.4, and return A_D plus the ultra-sparsifier of the Laplacian matrix A_L.

`BuildPreconditioners`(A_0),

1. $i = 0$.
2. Repeat
   (a) $i = i + 1$.
   (b) $B_i = \texttt{UltraSparsify}(A_{i-1}, k)$.
   (c) $(P_i, L_i, A_i) = \texttt{partialChol}(B_i)$.
   (d) Set Π_i to be the projection onto the span of B_i.

   Until A_i has dimension less than $66\chi + 6$.
3. Set $\ell = i$.
4. Compute $Z_\ell = {A_\ell}^\dagger$.

We now make a few observations about the sequence of matrices this algorithm generates. In the following, we let $\mathrm{noff}(A)$ denote the number of non-zero off-diagonal entries in the upper-triangular portion of A, and let $\dim(A)$ denote the dimension of A.

Proposition 5.1 (Recursive Preconditioning)**.** *If A_0 is a symmetric, irreducible,* SDDM_0*-matrix, and for each i the matrix B_i is a k-ultra-sparsifier of A_i, then*

(*a*) *For $i \geq 1$, $\mathrm{noff}(A_i) \leq (3\chi/k)\mathrm{noff}(A_{i-1})$.*

(*b*) *For $i \geq 1$, $\dim(A_i) \leq (2\chi/k)\mathrm{noff}(A_{i-1})$.*

(*c*) *For $i \geq 1$, $\dim(B_i) = \dim(A_{i-1})$.*

(*d*) *Each of B_i and A_i is an irreducible* SDDM_0*-matrix.*

(*e*) *Each A_i and B_i is a Laplacian matrix if and only if A_0 is as well.*

(*f*) *If A_0 is a Laplacian matrix, then each Π_i is a projection orthogonal to the all-1's vector. Otherwise, each Π_i is the identity.*

Proof. Let n_i be the dimension of A_i. Definition 1.1 tells us that

$$\mathrm{noff}(B_i) \leq n - 1 + h\mathrm{noff}(A_i)/k = n - 1 + \mathrm{noff}(A_i)\chi/k.$$

Parts (*a*), (*b*), (*d*) and (*e*) now follow from Proposition 4.1. Part (*c*) is obvious, and part (*f*) follows from Proposition 3.6. □

Our recursive solver will use each matrix B_i as a preconditioner for A_{i-1}. But rather than solve systems in B_i directly, it will reduce these to systems in A_i, which will in turn be solved recursively. Our solver will use the preconditioned Chebyshev method, instead of the preconditioned conjugate gradient. This choice is dictated by the requirements of our analysis rather than by common sense. Our preconditioned Chebyshev method will not take the preconditioner

B_i as input. Rather, it will take a subroutine $\texttt{solve}_{B_i}$ that produces approximate solutions to systems in B_i. So that we can guarantee that our solvers will be linear operators, we will fix the number of iterations that each will perform, as opposed to allowing them to terminate upon finding a sufficiently good solution. While this trick is necessary for our analysis, it may also be unnecessary in practice[1].

For concreteness, we present pseudocode for the variant of the preconditioned Chebyshev algorithm that we will use. It is a modification of the pseudocode presented in [BBC+94, page 36], the difference being that it takes as input a parameter t determining the number of iterations it executes (and some variable names have been changed).

$\boldsymbol{x} = \texttt{precondCheby}(A, \boldsymbol{b}, t, f(\cdot), \lambda_{min}, \lambda_{max})$
(0) Set $\boldsymbol{x} = \boldsymbol{0}$.
(1) $\boldsymbol{r} = \boldsymbol{b}$
(2) $d = (\lambda_{max} + \lambda_{min})/2$, $c = (\lambda_{max} - \lambda_{min})/2$
(3) for $i = 1, \ldots, t$,
  (a) $\boldsymbol{z} = f(\boldsymbol{r})$
  (b) if $i = 1$,
    $\boldsymbol{x} = \boldsymbol{z}$
    $\alpha = 2/d$
  else,
    $\beta = (c\alpha/2)^2$
    $\alpha = 1/(d - \beta)$
    $\boldsymbol{x} = \boldsymbol{z} + \beta\boldsymbol{x}$
  (c) $\boldsymbol{x} = \boldsymbol{x} + \alpha\boldsymbol{x}$
  (d) $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{x}$

Proposition 5.2 (Linear Chebyshev)**.** *Let A be a positive semi-definite matrix and f be a positive semi-definite, symmetric linear operator such that*

$$\lambda_{min} f^\dagger \preccurlyeq A \preccurlyeq \lambda_{max} f^\dagger. \tag{6}$$

Let $\epsilon < 1$ and let

$$t \geq \left\lceil \frac{1}{2}\sqrt{\frac{\lambda_{max}}{\lambda_{min}}} \ln \frac{2}{\epsilon} \right\rceil. \tag{7}$$

Then, the function $\texttt{precondCheby}(A, \boldsymbol{b}, t, f, \lambda_{min}, \lambda_{max})$ *is a symmetric linear operator in $\boldsymbol{b}$ with the same nullspace as A. Moreover, if Z is the matrix realizing this operator, then*

$$(1 - \epsilon)Z^\dagger \preccurlyeq A \preccurlyeq (1 + \epsilon)Z^\dagger.$$

[1] One could obtain a slightly weaker analysis of this algorithm if one instead allowed the Chebyshev solvers to terminate as soon as they found a sufficiently accurate solution. In an early version of this paper, we analyzed such an algorithm using the analysis of the inexact preconditioned Chebyshev iteration by Golub and Overton [GO88]. This analysis was improved by applying a slight extension by Joshi [Jos97] of Golub and Overton's analysis. The idea of bypassing these analysis by forcing our solvers to be linear operators was suggested to us by Vladimir Rokhlin.

Proof. By Proposition 3.1, condition (6) implies that f and A have the same nullspace. An inspection of the pseudo-code reveals that the function computed by `precondCheby` can be expressed as a sum of monomials of the form $f(Af)^i$, from which it follows that this function is a symmetric linear operator having the same nullspace as A. Let Z be the matrix realizing this operator.

Standard analyses of the preconditioned Chebyshev algorithm [Axe85, Section 5.3] imply that for all $\boldsymbol{b}$ in the range of A,

$$\left\| Z\boldsymbol{b} - A^\dagger \boldsymbol{b} \right\|_A \leq \epsilon \left\| A^\dagger \boldsymbol{b} \right\|_A .$$

Now, consider any non-zero eigenvalue λ and eigenvector $\boldsymbol{b}$ of AZ, so that

$$AZ\boldsymbol{b} = \lambda \boldsymbol{b}.$$

Multiplying on the left by $A^\dagger$ and using the fact that Z and A have the same nullspace, we obtain

$$Z\boldsymbol{b} = \lambda A^\dagger \boldsymbol{b}.$$

Plugging this into the previous inequality, we find

$$\epsilon \left\| A^\dagger \boldsymbol{b} \right\|_A \geq \left\| Z\boldsymbol{b} - A^\dagger \boldsymbol{b} \right\|_A = |\lambda - 1| \left\| A^\dagger \boldsymbol{b} \right\|_A ,$$

and so λ must lie between $1-\epsilon$ and $1+\epsilon$. Applying Proposition 3.3, we obtain

$$(1-\epsilon)Z^\dagger \preccurlyeq A \preccurlyeq (1+\epsilon)Z^\dagger.$$

□

We can now state the subroutine $\texttt{solve}_{B_i}$ for $i = 1, \ldots, \ell$.

$\boldsymbol{x} = \texttt{solve}_{B_i}(\boldsymbol{b})$

1. Set $\lambda_{min} = 1 - 2e^{-2}$, $\lambda_{max} = (1 + 2e^{-2})k$ and $t = \left\lceil 1.33\sqrt{k} \right\rceil$.
2. Set $\boldsymbol{s} = L_i^{-1} P_i^{-1} \Pi_i \boldsymbol{b}$.
3. Write $\boldsymbol{s} = \begin{pmatrix} \boldsymbol{s}_0 \\ \boldsymbol{s}_1 \end{pmatrix}$, where the dimension of $\boldsymbol{s}_1$ is the size of A_i.
4. Set $\boldsymbol{y}_0 = \boldsymbol{s}_0$, and
   (a) if $i = \ell$, set $\boldsymbol{y}_1 = Z_\ell \boldsymbol{s}_1$
   (b) else, set $\boldsymbol{y}_1 = \texttt{precondCheby}(A_i, \boldsymbol{s}_1, \texttt{solve}_{B_{i+1}}, t, \lambda_{min}, \lambda_{max})$.
5. Set $\boldsymbol{x} = \Pi_i P_i^{-T} L_i^{-T} \begin{pmatrix} \boldsymbol{y}_0 \\ \boldsymbol{y}_1 \end{pmatrix}$.

We have chosen the parameters λ_{min}, λ_{max}, and t so that inequality (7) holds for for $\epsilon = 2e^{-2}$.

We note that we apply L_i^{-T} and L_i^{-1} by forward and backward substitution, rather than by constructing the inverses. Similarly, Π_i may be applied in time proportional to the length of $\boldsymbol{b}$ as it is either the identity, or the operator that orthogonalizes with respect to the all-1's vector. We remark that the multiplications by Π_i are actually unnecessary in our code, as $\texttt{solve}_{B_i}$ will only appear inside a call to `precondCheby`, in which case it is multiplied on either side by matrices that implicitly contain Π_i. However, our analysis is simpler if we include these applications of Π_i.

Lemma 5.3 (Correctness of $\texttt{solve}_{B_i}$)**.** *If A is an irreducible* SDDM$_0$*-matrix and $B_i \preccurlyeq A_{i-1} \preccurlyeq kB_i$ for all $i \geq 1$, then for $1 \leq i \leq \ell$,*

(a) The function $\texttt{solve}_{B_i}$ is a symmetric linear operator.

(b) The function $\texttt{precondCheby}(A_{i-1}, \boldsymbol{b}, \texttt{solve}_{B_i}, t, \lambda_{min}, \lambda_{max})$ is a symmetric linear operator in $\boldsymbol{b}$.

(c)

$$(1 - 2e^{-2}){Z_i}^{\dagger} \preccurlyeq A_i \preccurlyeq (1 + 2e^{-2}){Z_i}^{\dagger},$$

where for $i \leq l - 1$, Z_i is the matrix such that

$$Z_i \boldsymbol{s}_1 = \texttt{precondCheby}(A_i, \boldsymbol{s}_1, \texttt{solve}_{B_{i+1}}, t, \lambda_{min}, \lambda_{max}).$$

(d)

$$(1 - 2e^{-2}){\texttt{solve}_{B_i}}^{\dagger} \preccurlyeq B_i \preccurlyeq (1 + 2e^{-2}){\texttt{solve}_{B_i}}^{\dagger}.$$

Proof. We first prove (a) and (b) by reverse induction on i. The base case of our induction is when $i = \ell$, in which case `BuildPreconditioners` sets $Z_\ell = {A_\ell}^{\dagger}$, and so

$$\texttt{solve}_{B_\ell} = \Pi_\ell P_\ell^{-T} L_\ell^{-T} \begin{pmatrix} I & 0 \\ 0 & Z_\ell \end{pmatrix} L_\ell^{-1} P_\ell^{-1} \Pi_\ell,$$

which is obviously a symmetric linear operator. Given that $\texttt{solve}_{B_i}$ is a symmetric linear operator, part (b) for A_{i-1} follows from Proposition 5.2. Given that (b) holds for A_i and that the call to `precondCheby` is realized by a symmetric matrix Z_i, we then have that

$$\texttt{solve}_{B_i} = \Pi_i P_i^{-T} L_i^{-T} \begin{pmatrix} I & 0 \\ 0 & Z_i \end{pmatrix} L_i^{-1} P_i^{-1} \Pi_i$$

is a symmetric linear operator. We may thereby establish that (a) and (b) hold for all $1 \geq i \geq \ell$.

We now prove properties (c) and (d), again by reverse induction. By construction $Z_\ell = {A_\ell}^{\dagger}$, so (c) holds for $i = \ell$. To see that if (c) holds for i, then (d) does also, note that

$$\begin{aligned}
(1 - 2e^{-2}){Z_i}^{\dagger} &\preccurlyeq A_i && \text{implies} \\
(1 - 2e^{-2}){A_i}^{\dagger} &\preccurlyeq Z_i, && \text{by Proposition 3.2, which implies} \\
(1 - 2e^{-2}) \begin{pmatrix} I & 0 \\ 0 & {A_i}^{\dagger} \end{pmatrix} &\preccurlyeq \begin{pmatrix} I & 0 \\ 0 & Z_i \end{pmatrix} && \text{which implies}
\end{aligned}$$

$$(1-2e^{-2})B_i{}^\dagger = (1-2e^{-2})\Pi_i P_i^{-T} L_i^{-T} \begin{pmatrix} I & 0 \\ 0 & A_i{}^\dagger \end{pmatrix} L_i^{-1} P_i^{-1} \Pi_i \qquad \text{(by Proposition 4.1 } (e)\text{)}$$
$$\preccurlyeq \Pi_i P_i^{-T} L_i^{-T} \begin{pmatrix} I & 0 \\ 0 & Z_i \end{pmatrix} L_i^{-1} P_i^{-1} \Pi_i = \texttt{solve}_{B_i},$$

which by Proposition 3.2 implies $(1-2e^{-2})\texttt{solve}_{B_i}{}^\dagger \preccurlyeq B_i$. The inequality $B_i \preccurlyeq (1+2e^{-2})\texttt{solve}_{B_i}{}^\dagger$ may be established similarly.

To show that when (d) holds for i then (c) holds for $i-1$, note that (d) and $B_i \preccurlyeq A_{i-1} \preccurlyeq k \cdot B_i$ imply

$$(1-2e^{-2})\texttt{solve}_{B_i}{}^\dagger \preccurlyeq A_{i-1} \preccurlyeq k(1+2e^{-2})\texttt{solve}_{B_i}{}^\dagger.$$

So, (c) for $i-1$ now follows from Proposition 5.2 and the fact that $\lambda_{min}, \lambda_{max}$ and t have been chosen so that inequality (7) is satisfied with $\epsilon = 2e^{-2}$. □

Lemma 5.4 (Complexity of $\texttt{solve}_{B_i}$)**.** *If A_0 is an irreducible, n-by-n SDDM_0-matrix with $2m$ non-zero off-diagonal entries and each B_i is a k-ultra-sparsifier of A_{i-1}, then $\texttt{solve}_{B_1}$ runs in time*

$$O(n+m).$$

Proof. Let T_i denote the running time of $\texttt{solve}_{B_i}$. We will prove by reverse induction on i that there exists a constant c such that

$$T_i \leq c\left(\dim(B_i) + (\gamma\chi+\delta)(\mathrm{noff}(A_i) + \dim(A_i))\right), \tag{8}$$

where

$$\gamma = 196 \quad \text{and} \quad \delta = 15.$$

This will prove the lemma as $\dim(B_1) = \dim(A_0) = n$, and Proposition 5.1 implies

$$(\gamma\chi+\delta)(\mathrm{noff}(A_i) + \dim(A_i)) \leq (\gamma\chi+\delta)\frac{5\chi m}{k} \leq m\frac{5\gamma\chi^2 + 5\delta\chi}{(14\chi+1)^2} = O(m).$$

To prove (8), we note that there exists a constant c so that steps 2 and 5 take time at most $c(\dim(B_i))$ (by Proposition 4.1), step 4a takes time at most $c(\dim(A_\ell)^2)$, and step 4b takes time at most $t(c\cdot\dim(A_i) + c\cdot\mathrm{noff}(A_i) + T_{i+1})$, where t is as defined on step 1 of $\texttt{solve}_{B_i}$.

The base case of our induction will be $i = \ell$, in which case the preceding analysis implies

$$\begin{aligned} T_\ell &\leq c\left(\dim(B_\ell) + \dim(A_\ell)^2\right) \\ &\leq c\left(\dim(B_\ell) + (66\chi+6)\dim(A_\ell)\right), \qquad \text{(by step 2 of } \texttt{BuildPreconditioners}\text{)} \end{aligned}$$

which satisfies (8). We now prove (8) is true for $i < \ell$, assuming it is true for $i+1$. We have

$$\begin{aligned} T_i &\leq c\left(\dim(B_i)\right) + t(c\cdot\dim(A_i) + c\cdot\mathrm{noff}(A_i) + T_{i+1}) \\ &\leq c\left[\dim(B_i) + t\big(\dim(A_i) + \mathrm{noff}(A_i) + \dim(B_{i+1}) + (\gamma\chi+\delta)(\mathrm{noff}(A_{i+1}) + \dim(A_{i+1}))\big)\right] \end{aligned}$$

(by the induction hypothesis)

$$\leq c\left[\dim(B_i) + t\big(2\,\dim(A_i) + \mathrm{noff}(A_i) + (\gamma\chi+\delta)(5\,\mathrm{noff}(A_i)\,\chi/k)\big)\right]$$

(by Proposition 5.1)

$$\leq c\left[\dim\left(B_i\right) + t\left(2\ \dim\left(A_i\right) + 6\ \text{noff}\left(A_i\right)\right)\right],$$

as $\gamma\chi^2 + \delta\chi \leq k$. As

$$6t \leq 6 \cdot (1.33(14\chi + 1) + 1) \leq \gamma\chi + \delta,$$

we have proved that (8) is true for i as well. □

We now state and analyze our ultimate solver.

$\boldsymbol{x} = \texttt{solve}(A, \boldsymbol{b}, \epsilon)$

1. Set $\lambda_{min} = 1 - 2e^{-2}$, $\lambda_{max} = (1 + 2e^{-2})k$ and $t = \left\lceil 0.67\sqrt{k}\ln(2/\epsilon)\right\rceil$.
2. Run $\texttt{BuildPreconditioners}(A)$.
3. $\boldsymbol{x} = \texttt{precondCheby}(A, \boldsymbol{b}, \texttt{solve}_{B_1}, t, \lambda_{min}, \lambda_{max})$.

Theorem 5.5 (Nearly Linear-Time Solver)**.** *On input an irreducible n-by-n SDDM_0-matrix A with $2m$ non-zero off-diagonal entries and an n-vector $\boldsymbol{b}$, with probability at least $1 - 1/500$, $\texttt{solve}(A, \boldsymbol{b}, \epsilon)$ runs in time*

$$O(m\log^{c_4} m\log(1/\epsilon)) + m\log^{O(1)} m$$

and produces an $\tilde{\boldsymbol{x}}$ satisfying

$$\left\|\tilde{\boldsymbol{x}} - A^{\dagger}\boldsymbol{b}\right\|_A \leq \epsilon\left\|A^{\dagger}\boldsymbol{b}\right\|_A.$$

Proof. By Proposition 5.1, the numbers $\text{noff}\left(A_i\right)$ are geometrically decreasing, and $l \leq \log_{k/3\chi} m$. So we may use Theorem 10.5 to show that the time required to build the preconditioners is at most $m\log^{O(1)} m$. If each B_i is a k-ultra-sparsifier of A_{i-1}, then the bound on the A-norm of the output follows by an analysis similar to that used to prove Lemma 5.3. In this case, we may use Lemma 5.4 to bound on the running time of step 3 by

$$O\left(mt\right) = O(m\sqrt{k}\log(1/\epsilon)) = O\left(m\log^{c_4} n\log(1/\epsilon)\right).$$

The probability that there is some B_i that is not a k-ultra-sparsifier of A_{i-1} is at most

$$\sum_i \frac{1}{2\ \dim\left(B_i\right)} \leq \frac{l}{2(66\chi + 6)} \leq \frac{\log_{k/3\chi} m}{2(66\chi + 6)} < 1/500,$$

assuming $c_3, c_4 \geq 1$. □

If the non-zero structure of A is planar, then by Theorem 9.5, we can replace all the calls to $\texttt{UltraSparsify}$ in the above algorithm with calls to $\texttt{UltraSimple}$. By Theorem 9.1, this is like having (k, h)-ultra-sparsifiers with $h = O(\log n\log^2\log n)$. Thus, the same analysis goes through with $\chi = O(\log n\log^2\log n)$, and the resulting linear system solver runs in time

$$O(n\log^2 n + n\log n\ \log^2\log n\ \log(1/\epsilon)).$$

We remark that our analysis is very loose when m is much larger than n. In this case, the first ultra-sparsifier constructed, B_1, will probably have close to n edges, which could be much lower than the bound proved in Proposition 5.1. While it is not necessary for the proof of our theorem, one could remove this slack by setting $B_1 = \texttt{Sparsify}(A_0, 1/2n)$ in this case.

6 Computing Approximate Fiedler Vectors

Fiedler [Fie73] was the first to recognize that the eigenvector associated with the second-smallest eigenvalue of the Laplacian matrix of a graph could be used to partition a graph. From a result of Mihail [Mih89], we know that any vector whose Rayleigh quotient is close to this eigenvalue can also be used to find a good partition. We call such a vector an *approximate Fiedler vector*.

Definition 6.1 (Approximate Fiedler Vector)**.** *For a Laplacian matrix A, $\boldsymbol{v}$ is an ϵ-approximate Fiedler vector if $\boldsymbol{v}$ is orthogonal to the all-1's vector and*

$$\frac{\boldsymbol{v}^T A \boldsymbol{v}}{\boldsymbol{v}^T \boldsymbol{v}} \leq (1+\epsilon)\lambda_2(A),$$

where $\lambda_2(A)$ is the second-smallest eigenvalue of A.

Our linear system solvers may be used to quickly compute ϵ-approximate Fiedler vectors. We will prove that the following algorithm does so with probability at least $1-p$.

$\boldsymbol{v} = \texttt{ApproxFiedler}(A, \epsilon, p)$

1. Set $\lambda_{min} = 1 - 2e^{-2}$, $\lambda_{max} = (1+2e^{-2})k$ and $t = \left\lceil 0.67\sqrt{k}\ln(8/\epsilon) \right\rceil$.
2. Set $k = 8\ln(18(n-1)/\epsilon)/\epsilon$.
3. For $a = 1, \ldots, \lceil \log_2 p \rceil$.
   (a) Run $\texttt{BuildPreconditioners}(A)$.
   (b) Choose $\boldsymbol{r}^0$ to be a random unit vector orthogonal to the all-1's vector.
   (c) For $b = 1, \ldots, k$
   $$\boldsymbol{r}^b = \texttt{precondCheby}(A, \boldsymbol{r}^{b-1}, \texttt{solve}_{B_1}, t, \lambda_{min}, \lambda_{max}).$$
   (d) Set $\boldsymbol{v}_a = \boldsymbol{r}^k$.
4. Let a_0 be the index of the vector minimizing $\boldsymbol{v}_{a_0}^T A \boldsymbol{v}_{a_0} / \boldsymbol{v}_{a_0}^T \boldsymbol{v}_{a_0}$.
5. Set $\boldsymbol{v} = \boldsymbol{v}_{a_0}$.

Theorem 6.2. *On input a Laplacian matrix A with m non-zero entries and $\epsilon, p > 0$, with probability at least $1 - 1/p$, $\texttt{ApproxFiedler}(A, \epsilon, p)$ computes an ϵ-approximate Fiedler vector of A in time*

$$m \log^{O(1)} m \log(1/p) \log(1/\epsilon)/\epsilon.$$

Our proof of Theorem 6.2 will use the following proposition.

Proposition 6.3. *If Z is a matrix such that*

$$(1-\epsilon)Z^\dagger \preccurlyeq A \preccurlyeq (1+\epsilon)Z^\dagger,$$

and $\boldsymbol{v}$ is a vector such that $\boldsymbol{v}^T Z^\dagger \boldsymbol{v} \leq (1+\epsilon)\lambda_2(Z^\dagger)$, for some $\epsilon \leq 1/5$, then $\boldsymbol{v}$ is a 4ϵ-approximate Fiedler vector of A.

Proof. We first observe that

$$\lambda_2(Z^\dagger) \leq \lambda_2(A)/(1-\epsilon).$$

We then compute

$$\begin{aligned}
\boldsymbol{v}^T A \boldsymbol{v} &\leq (1+\epsilon)\boldsymbol{v}^T Z^\dagger \boldsymbol{v} \\
&\leq (1+\epsilon)(1+\epsilon)\lambda_2(Z^\dagger) \\
&\leq (1+\epsilon)(1+\epsilon)\lambda_2(A)/(1-\epsilon) \\
&\leq (1+4\epsilon)\lambda_2(A),
\end{aligned}$$

for $\epsilon \leq 1/5$. □

Proof of Theorem 6.2. As we did in the proof of Lemma 5.3 and Theorem 5.5, we can show that $\texttt{precondCheby}(A, \boldsymbol{b}, \texttt{solve}_{B_1}, t, \lambda_{min}, \lambda_{max})$ is a linear operator in $\boldsymbol{b}$. Let Z denote the matrix realizing this operator. As in the proof of Lemma 5.3, we can show that $(1-\epsilon/4)Z^\dagger \preccurlyeq A \preccurlyeq (1+\epsilon/4)Z^\dagger$.

By Proposition 6.3, it suffices to show that with probability at least 1/2 each vector $\boldsymbol{v}_a$ satisfies

$$\boldsymbol{v}_a^T Z^\dagger \boldsymbol{v}_a / \boldsymbol{v}_a^T \boldsymbol{v}_a \leq (1+\epsilon/4)\lambda_2(Z^\dagger).$$

To this end, let $0 = \mu_1 \leq \mu_2 \leq \cdots \leq \mu_n$ be the eigenvalues of $Z^\dagger$, and let $\mathbf{1} = \boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ be corresponding eigenvectors. Let

$$\boldsymbol{r}^0 = \sum_{i \geq 2} \alpha_i \boldsymbol{u}_i,$$

and recall that (see *e.g.* [SST06, Lemma B.1])

$$\Pr\left[|\alpha_2| \geq 2/3\sqrt{(n-1)}\right] \geq \frac{2}{\sqrt{2\pi}} \int_{2/3}^{\infty} e^{-t^2/2}\, dt \geq 0.504.$$

Thus, with probably at least 1/2, the call to `BuildPreconditioners` succeeds and $|\alpha_2| \geq 2/3\sqrt{(n-1)}$. In this case,

$$k \geq 8 \ln(8/\alpha_2^2 \epsilon)/\epsilon. \tag{9}$$

We now show that this inequality implies that $\boldsymbol{r}^k$ satisfies

$$\frac{(\boldsymbol{r}^k)^T Z^\dagger \boldsymbol{r}^k}{(\boldsymbol{r}^k)^T \boldsymbol{r}^k} \leq (1+\epsilon/4)\mu_2.$$

To see this, let j be the greatest index such that $\mu_j \leq (1+\epsilon/8)\mu_2$, and compute

$$\boldsymbol{r}^k = Z^k \boldsymbol{r}^0 = \sum_{i \geq 2} \alpha_i / \mu_i^k \boldsymbol{u}_i,$$

so

$$\begin{aligned}
\frac{(\boldsymbol{r}^k)^T Z^\dagger \boldsymbol{r}^k}{(\boldsymbol{r}^k)^T \boldsymbol{r}^k} &= \frac{\sum_{i\geq 2} \alpha_i^2/\mu_i^{2k-1}}{\sum_{i\geq 2} \alpha_i^2/\mu_i^{2k}} \\
&\leq \frac{\sum_{j\geq i\geq 2} \alpha_i^2/\mu_i^{2k-1}}{\sum_{j\geq i\geq 2} \alpha_i^2/\mu_i^{2k}} + \frac{\sum_{i>j} \alpha_i^2/\mu_i^{2k-1}}{\sum_{i\geq 2} \alpha_i^2/\mu_i^{2k}} \\
&\leq \mu_j + \frac{\sum_{i>j} \alpha_i^2/\mu_i^{2k-1}}{\alpha_2^2/\mu_2^{2k}} \\
&\leq (1+\epsilon/8)\mu_2 + \mu_2 \left(\frac{\sum_{i>j} \alpha_i^2 (\mu_2/\mu_i)^{2k-1}}{\alpha_2^2} \right) \\
&\leq (1+\epsilon/8)\mu_2 + \mu_2 \left(\frac{\sum_{i>j} \alpha_i^2 (1/(1+\epsilon/8))^{2k-1}}{\alpha_2^2} \right) \\
&\leq (1+\epsilon/8)\mu_2 + \mu_2 \sum_{i>j} \alpha_i^2 \epsilon/8 \qquad \text{(by inequality (9))} \\
&\leq (1+\epsilon/8)\mu_2 + \mu_2(\epsilon/8) \\
&\leq (1+\epsilon/4)\mu_2.
\end{aligned}$$

□

7 Laplacians and Weighted Graphs

We will find it convenient to describe and analyze our preconditioners for Laplacian matrices in terms of weighted graphs. This is possible because of the isomorphism between Laplacian matrices and weighted graphs. To an n-by-n Laplacian matrix A, we associate the graph with vertex set $\{1, \ldots, n\}$ having an edge between vertices u and v of weight $-A(u,v)$ for each u and v such that $A(u,v)$ is non-zero.

All the graphs we consider in this paper will be weighed. If u and v are distinct vertices in a graph, we write $\mathbf{(}u,v\mathbf{)}$ to denote an edge between u and v of weight 1. Similarly, if $w > 0$, then we write $w\mathbf{(}u,v\mathbf{)}$ to denote an edge between u and v of weight w. A weighted graph is then a pair $G = (V, E)$ where V is a set of vertices and E is a set of weighted edges on V, each of which spans a distinct pair of vertices. The Laplacian matrix L_G of the graph G is the matrix such that

$$L_G(u,v) = \begin{cases} -w & \text{if there is an edge } w\mathbf{(}u,v\mathbf{)} \in E \\ 0 & \text{if } u \neq v \text{ and there is no edge between } u \text{ and } v \text{ in } E \\ \sum_{w\mathbf{(}u,x\mathbf{)}\in E} w & \text{if } u = v. \end{cases}$$

We recall that for every vector $\boldsymbol{x} \in \mathbb{R}^n$,

$$\boldsymbol{x}^T L_G \boldsymbol{x} = \sum_{w\mathbf{(}u,v\mathbf{)}\in E} w(\boldsymbol{x}_u - \boldsymbol{x}_v)^2.$$

For graphs G and H, we define the graph $G + H$ to be the graph whose Laplacian matrix is $L_G + L_H$.

8 Graphic Inequalities, Resistance, and Low-Stretch Spanning Trees

In this section, we introduce the machinery of "graphic inequalities" that underlies the proofs in the rest of the paper. We then introduce low-stretch spanning trees, and use graphic inequalities to bound how well a low-stretch spanning tree preconditions a graph. This proof provides the motivation for the construction in the next section.

We begin by overloading the notation $\preccurlyeq$ by writing

$$G \preccurlyeq H \qquad \text{and} \qquad E \preccurlyeq F$$

if $G = (V, E)$ and $H = (V, F)$ are two graphs such that their Laplacian matrices, L_G and L_H satisfy

$$L_G \preccurlyeq L_H.$$

Many facts that have been used in the chain of work related to this paper can be simply expressed with this notation. For example, the Splitting Lemma of [BGH+06] becomes

$$A_1 \preccurlyeq B_1 \quad \text{and} \quad A_2 \preccurlyeq B_2 \quad \text{implies} \quad A_1 + A_2 \preccurlyeq B_1 + B_2.$$

We also observe that if B is a subgraph of A, then

$$B \preccurlyeq A.$$

We define the *resistance* of an edge to be the reciprocal of its weight. Similarly, we define the resistance of a simple path to be the sum of the resistances of its edges. For example, the resistance of the path $w_1(1,2), w_2(2,3), w_3(3,4)$ is $(1/w_1 + 1/w_2 + 1/w_3)$. Of course, the resistance of a trivial path with one vertex and no edges is zero. If one multiplies all the weights of the edges in a path by α, its resistance decreases by a factor of α.

The next lemma says that a path of resistance r supports an edge of resistance r. This lemma may be derived from the Rank-One Support Lemma of [BH03b], and appears in simpler form as the Congestion-Dilation Lemma of [BGH+06] and Lemma 4.6 of [Gre96].

Lemma 8.1 (Path Inequality)**.** *Let $e = w(u, v)$ and let P be a path from u to v. Then,*

$$e \preccurlyeq w \ \textit{resistance}(P) \cdot P.$$

Proof. After dividing both sides by w, it suffices to consider the case $w = 1$. Without loss of generality, we may assume that $e = (1, k+1)$ and that P consists of the edges $w_i(i, i+1)$ for $1 \le i \le k$. In this notation, the lemma is equivalent to

$$(1, k+1) \preccurlyeq \left(\sum_i \frac{1}{w_i} \right) \left(w_1(1,2) + w_2(2,3) + \cdots + w_k(k, k+1) \right).$$

We prove this for the case $k = 2$. The general case follows by induction.

Recall Cauchy's inequality, which says that for all $0 < \alpha < 1$,

$$(a+b)^2 \le a^2/\alpha + b^2/(1-\alpha).$$

For $k = 2$, the lemma is equivalent to

$$(x_1 - x_3)^2 \leq (1 + w_1/w_2)(x_1 - x_2)^2 + (1 + w_2/w_1)(x_2 - x_3)^2,$$

which follows from Cauchy's inequality with $\alpha = w_2/(w_1 + w_2)$. □

Recall that a *spanning tree* of a weighted graph $G = (V, E)$ is a connected subgraph of G with exactly $|V| - 1$ edges. The weights of edges that appear in a spanning tree are assumed to be the same as in G. If T is a spanning tree of a graph $G = (V, E)$, then for every pair of vertices $u, v \in V$, T contains a unique path from u to v. We let $T(u, v)$ denote this path. We now use graphic inequalities to derive a bound on how well T preconditions G. This bound strengthens a bound of Boman and Hendrickson [BH03b, Lemma 4.9].

Lemma 8.2. *Let $G = (V, E)$ be a graph and let T be a spanning tree of G. Then,*

$$T \preccurlyeq G \preccurlyeq \left(\sum_{e \in E} \frac{\mathit{resistance}(T(e))}{\mathit{resistance}(e)} \right) \cdot T.$$

Proof. As T is a subgraph of G, $T \preccurlyeq G$ is immediate. To prove the right-hand inequality, we compute

$$\begin{aligned}
E &= \sum_{e \in E} e \\
&\preccurlyeq \sum_{e \in E} \frac{\text{resistance}(T(e))}{\text{resistance}(e)} \cdot T(e), && \text{by Lemma 8.1} \\
&\preccurlyeq \left(\sum_{e \in E} \frac{\text{resistance}(T(e))}{\text{resistance}(e)} \right) \cdot T, && \text{as } T(e) \preccurlyeq T.
\end{aligned}$$

□

Definition 8.3. *Given a tree T spanning a set of vertices V and a weighted edge $e = w(u, v)$ with $u, v \in V$, we define the stretch of e with respect to T to be*

$$\mathit{st}_T(e) = \frac{\mathit{resistance}(T(e))}{\mathit{resistance}(e)} = w \cdot \mathit{resistance}(T(e)).$$

If E is a set of edges on V, then we define

$$\mathit{st}_T(E) = \sum_{e \in E} \mathit{st}_T(e).$$

With this definition, the statement of Lemma 8.2 may be simplified to

$$T \preccurlyeq G \preccurlyeq \text{st}_T(E) \cdot T. \tag{10}$$

We will often use the following related inequality, which follows immediately from Lemma 8.1 and the definition of stretch.

$$w(u, v) \preccurlyeq \text{st}_T(w(u, v))\ T(u, v) = w\ \text{st}_T((u, v))\ T(u, v), \tag{11}$$

where we recall that $T(u, v)$ is the unique path in T from u to v.

9 Preconditioning with Low-Stretch Trees

In this section, we present a simple preconditioning algorithm, `UltraSimple`, that works by simply adding edges to low-stretch spanning trees. This algorithm is sufficient to obtain all our results for planar graphs. For arbitrary graphs, this algorithm might add too many additional edges. We will show in Section 10 how these extra edges can be removed via sparsification.

9.1 Low-Stretch Trees

Low-stretch spanning trees were introduced by Alon, Karp, Peleg and West [AKPW95]. At present, the construction of spanning trees with the lowest stretch is due to Abraham, Bartal and Neiman [ABN08], who prove

Theorem 9.1 (Low Stretch Spanning Trees)**.** *There exists an $O(m \log n + n \log^2 n)$-time algorithm, `LowStretch`, that on input a weighted connected graph $G = (V, E)$, outputs a spanning tree T of G such that*

$$st_T(E) \leq c_{ABN}\ m \log n \log\log n (\log\log\log n)^3,$$

where $m = |E|$, for some constant c_{ABN}. In particular, $st_T(E) = O(m \log n\ \log^2 \log n)$.

9.2 Augmenting Low-Stretch Spanning Trees

To decide which edges to add to the tree, we first decompose the tree into a collection of subtrees so that no non-singleton subtree is attached to too many edges of E of high stretch. In the decomposition, we allow subtrees to overlap at a single vertex, or even consist of just a single vertex. Then, for every pair of subtrees connected by edges of E, we add one such edge of E to the tree. The subtrees are specified by the subset of the vertices that they span.

Definition 9.2. *Given a tree T that spans a set of vertices V, a T-decomposition is a decomposition of V into sets $W_1, \ldots, W_h$ such that $V = \cup W_i$, the graph induced by T on each W_i is a tree, possibly with just one vertex, and for all $i \neq j$, $|W_i \cap W_j| \leq 1$.*

Given an additional set of edges E on V, a (T, E)-decomposition is a pair $(\{W_1, \ldots, W_h\}, \rho)$ where $\{W_1, \ldots, W_h\}$ is a T-decomposition and ρ is a map that sends each edge of E to a set or pair of sets in $\{W_1, \ldots, W_h\}$ so that for each edge in $(u, v) \in E$,

(a) if $\rho(u, v) = \{W_i\}$ then $\{u, v\} \in W_i$, and

(b) if $\rho(u, v) = \{W_i, W_j\}$, then either $u \in W_i$ and $v \in W_j$, or $u \in W_j$ and $v \in W_i$.

We remark that as the sets W_i and W_j can overlap, it is possible that $\rho(u, v) = \{W_i, W_j\}$, $u \in W_i$ and $v \in W_i \cap W_j$.

We use the following tree decomposition theorem to show that one can always quickly find a T-decomposition of E with few components in which the sum of stretches of the edges attached to each component is not too big. As the theorem holds for any non-negative function η on the edges, not just stretch, we state it in this general form.

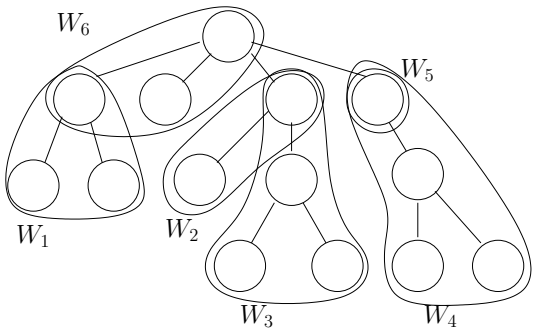

Figure 2: An example of a tree decomposition. Note that sets W_1 and W_6 overlap, and that set W_5 is a singleton set and that it overlaps W_4.

Theorem 9.3 (`decompose`)**.** *There exists a linear-time algorithm, which we invoke with the syntax*

$$(\{W_1, \ldots, W_h\}, \rho) = \texttt{decompose}(T, E, \eta, t),$$

that on input a set of edges E on a vertex set V, a spanning tree T on V, a function $\eta : E \to \mathbb{R}^+$, and an integer $1 < t \leq \sum_{e \in E} \eta(e)$, outputs a (T, E)-decomposition $(\{W_1, \ldots, W_h\}, \rho)$, such that

(a) $h \leq t$,

(b) for all W_i such that $|W_i| > 1$,

$$\sum_{e \in E : W_i \in \rho(e)} \eta(e) \leq \frac{4}{t} \sum_{e \in E} \eta(e).$$

For pseudo-code and a proof of this theorem, see Appendix C. We remark that when $t \geq n$, the algorithm can just construct a singleton set for every vertex.

For technical reasons, edges with stretch less than 1 can be inconvenient. So, we define

$$\eta(e) = \max(\mathrm{st}_T(e), 1) \quad \text{and} \quad \eta(E) = \sum_{e \in E} \eta(e). \tag{12}$$

The tree T should always be clear from context.

Given a (T, E)-decomposition, $(\{W_1, \ldots, W_h\}, \rho)$, we define the map

$$\sigma : \{1, \ldots, h\} \times \{1, \ldots, h\} \to E \cup \{\textit{undefined}\}$$

by setting

$$\sigma(i, j) = \begin{cases} \arg\max_{e : \rho(e) = \{W_i, W_j\}} \mathrm{weight}(e)/\eta(e), & \text{if } i \neq j \text{ and such an } e \text{ exists} \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{13}$$

In the event of a tie, we let e be the lexicographically least edge maximizing $\mathrm{weight}(e)/\eta(e)$ such that $\rho(e) = \{W_i, W_j\}$. Note that $\sigma(i, j)$ is a weighted edge.

The map σ tells us which edge from E between W_i and W_j to add to T. The following property of σ, which follows immediately from its definition, will be used in our analysis in this and the next section.

Proposition 9.4. *For every i, j such that $\sigma(i,j)$ is defined and for every $e \in E$ such that $\rho(e) = \{W_i, W_j\}$,*

$$\frac{\text{weight}(e)}{\eta(e)} \leq \frac{\text{weight}(\sigma(i,j))}{\eta(\sigma(i,j))}.$$

We can now state the procedure by which we augment a spanning tree.

$F = \texttt{AugmentTree}(T, E, t)$,
E is set of weighted edges,
T is a spanning tree of the vertices underlying E,
t is an integer.

1. Compute $\text{st}_T(e)$ for each edge $e \in E$.
2. Set $((W_1, \ldots, W_h), \rho) = \texttt{decompose}(T, E, \eta, t)$, where $\eta(e)$ is as defined in (12).
3. Set F to be the union of the weighted edges $\sigma(i,j)$ over all pairs $1 \leq i < j \leq h$ for which $\sigma(i,j)$ is defined, where $\sigma(i,j)$ is as defined in (13).

$A = \texttt{UltraSimple}(E, t)$

1. Set $T = \texttt{LowStretch}(E)$.
2. Set $F = \texttt{AugmentTree}(T, E, t)$.
3. Set $A = T \cup F$.

We remark that when $t \geq n$, `UltraSimple` can just return $A = E$.

Theorem 9.5 (`AugmentTree`)**.** *On input a set of weighted edges E, a spanning subtree T, and an integer $1 < t \leq \eta(E)$, the algorithm* ***`AugmentTree`*** *runs in time $O(m \log n)$, where $m = |E|$. The set of edges F output by the algorithm satisfies*

(a) $F \subseteq E$,

(b) $|F| \leq t^2/2$,

(c) If $T \subseteq E$, as happens when ***`AugmentTree`*** *is called by* ***`UltraSimple`****, then $(T \cup F) \preccurlyeq E$.*

(d)

$$E \preccurlyeq \frac{12\eta(E)}{t} \cdot (T \cup F). \tag{14}$$

Moreover, if E is planar then A is planar and $|F| \leq 3t - 6$.

Proof. In Appendix B, we present an algorithm for computing the stretch of each edge of E in time $O(m \log n)$. The remainder of the analysis of the running time is trivial. Part (a) follows immediately from the statement of the algorithm. When $T \subseteq E$, $T \cup F \subseteq E$, so part (c) follows as well.

To verify (b), note that the algorithm adds at most one edge to F for each pair of sets in $W_1, \ldots, W_h$, and there are at most $\binom{t}{2} \leq t^2/2$ such pairs. If E is planar, then F must be planar as F is a subgraph of E. Moreover, we can use Lemma C.1 to show that the graph induced by E on the sets $W_1, \ldots, W_h$ is also planar. Thus, the number of pairs of these sets connected by edges of E is at most the maximum number of edges in a planar graph with t vertices, $3t - 6$.

We now turn to the proof of part (d). Set

$$\beta = 4\eta(E)/t. \tag{15}$$

By Theorem 9.3, ρ and $W_1, \ldots, W_h$ satisfy

$$\sum_{e: W_i \in \rho(e)} \eta(e) \leq \beta, \quad \text{for all } W_i \text{ such that } |W_i| > 1. \tag{16}$$

Let E_i^{int} denote the set of edges e with $\rho(e) = (W_i, W_i)$, and let E_i^{ext} denote the set of edges e with $|\rho(e)| = 2$ and $W_i \in \rho(e)$. Let $E^{int} = \cup_i E_i^{int}$ and $E^{ext} = \cup_i E_i^{ext}$. Also, let T_i denote the tree formed by the edges of T inside the set W_i. Note that when $|W_i| = 1$, T_i and E_i^{int} are empty.

We will begin by proving that when $|W_i| > 1$,

$$E_i^{int} \preccurlyeq \left(\sum_{e \in E_i^{int}} \eta(e) \right) T_i, \tag{17}$$

from which it follows that

$$E^{int} \preccurlyeq \sum_{i: |W_i| > 1} \left(\sum_{e \in E_i^{int}} \eta(e) \right) T_i. \tag{18}$$

For any edge $e \in E_i^{int}$, the path in T between the endpoints of e lies entirely in T_i. So, by (11) we have

$$e \preccurlyeq \mathrm{st}_T(e) \cdot T_i \preccurlyeq \eta(e) \cdot T_i.$$

Inequality (17) now follows by summing over the edges $e \in E_i^{int}$.

We now define the map $\tau : E \to E \cup \{\mathit{undefined}\}$ by

$$\tau(e) = \begin{cases} \sigma(i,j), & \text{if } |\rho(e)| = 2, \text{ where } \rho(e) = \{W_i, W_j\}, \text{ and} \\ \mathit{undefined} & \text{otherwise.} \end{cases} \tag{19}$$

To handle the edges bridging components, we prove that for each edge e with $\rho(e) = (W_i, W_j)$,

$$e \preccurlyeq 3\eta(e)(T_i + T_j) + 3\frac{\mathrm{weight}(e)}{\mathrm{weight}(\tau(e))} \cdot \tau(e) \tag{20}$$

Let $e = w\big(u, v\big)$ be such an edge, with $u \in W_i$ and $v \in W_j$. Let $\tau(e) = z\big(x, y\big)$, with $x \in W_i$ and $y \in W_j$. Let t_i denote the last vertex in T_i on the path in T from u to v (see Figure 3). If T_i is empty, $t_i = u$. Note that t_i is also the last vertex in T_i on the path in T from x to y. Define t_j similarly. As $T_i(u, x) \subseteq T_i(u, t_i) \cup T_i(t_i, x)$, the tree T_i contains a path from u to x of resistance

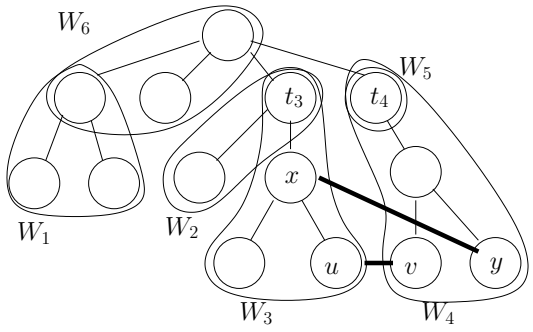

Figure 3: In this example, $e = w\big(u, v\big)$ and $\tau(e) = z\big(x, y\big)$.

at most

$$\text{resistance}(T_i(u, t_i)) + \text{resistance}(T_i(t_i, x)),$$

and the tree T_j contains a path from y to v of resistance at most

$$\text{resistance}(T_j(y, t_j)) + \text{resistance}(T_j(t_j, v)).$$

Furthermore, as $T_i(u, t_i) + T_j(t_j, v) \subseteq T(u, v)$ and $T_i(t_i, x) + T_j(y, t_j) \subseteq T(x, y)$, the sum of the resistances of the paths from u to x in T_i and from y to v in T_j is at most

$$\begin{aligned}\text{resistance}(T(u, v)) + \text{resistance}(T(x, y)) &= \text{st}_T(e)/w + \text{st}_T(\tau(e))/z \\ &\leq \eta(e)/w + \eta(\tau(e))/z \\ &\leq 2\eta(e)/w,\end{aligned}$$

where the last inequality follows from Proposition 9.4. Thus, the graph

$$3\eta(e)(T_i + T_j) + 3w\big(x, y\big) = 3\eta(e)(T_i + T_j) + 3\frac{\text{weight}(e)}{\text{weight}(\tau(e))} \cdot \tau(e)$$

contains a path from u to v of resistance at most

$$\frac{2}{3}\frac{1}{w} + \frac{1}{3}\frac{1}{w} = \frac{1}{w},$$

which by Lemma 8.1 implies (20).

We will now sum (20) over every edge $e \in E_i^{ext}$ for every i, observing that this counts every

edge in E^{ext} twice.

$$\begin{aligned}
E^{ext} &= (1/2)\sum_i \sum_{e\in E_i^{ext}} e \\
&\preccurlyeq \sum_i \sum_{e\in E_i^{ext}} 3\eta(e)T_i + (1/2)\sum_i \sum_{e\in E_i^{ext}} 3\frac{\text{weight}(e)}{\text{weight}(\tau(e))}\cdot\tau(e) \\
&= 3\sum_i \left(\sum_{e\in E_i^{ext}} \eta(e)\right) T_i + 3\sum_{e\in E^{ext}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))}\cdot\tau(e) \\
&= 3\sum_{i:|W_i|>1} \left(\sum_{e\in E_i^{ext}} \eta(e)\right) T_i + 3\sum_{e\in E^{ext}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))}\cdot\tau(e), \qquad (21)
\end{aligned}$$

as T_i is empty when $|W_i| = 1$.

We will now upper bound the right-hand side of (21). To handle boundary cases, we divide E^{ext} into two sets. We let E^{ext}_{single} consist of those $e \in E^{ext}$ for which both sets in $\rho(e)$ have size 1. We let $E^{ext}_{general} = E^{ext} - E^{ext}_{single}$ contain the rest of the edges in E^{ext}. For $e \in E^{ext}_{single}$, $\tau(e) = e$, while for $e \in E^{ext}_{general}$, $\tau(e) \in E^{ext}_{general}$.

For E^{ext}_{single}, we have

$$\sum_{e\in E^{ext}_{single}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))}\cdot\tau(e) = \sum_{e\in E^{ext}_{single}} \tau(e) = E^{ext}_{single}.$$

To evaluate the sum over the edges $e \in E^{ext}_{general}$, consider any $f \in E^{ext}_{general}$ in the image of τ. Let i be such that $f \in E^{ext}_i$ and $|W_i| > 1$. Then, for every e such that $\tau(e) = f$, we have $e \in E^{ext}_i$. So, by Proposition 9.4,

$$\begin{aligned}
\sum_{\substack{e\in E^{ext}\\ \tau(e)=f}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} &= \sum_{\substack{e\in E^{ext}_i\\ \tau(e)=f}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} \\
&\le \sum_{e\in E^{ext}_i} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} \le \sum_{e\in E^{ext}_i} \frac{\eta(e)}{\eta(\tau(e))} \le \sum_{e\in E^{ext}_i} \eta(e) \le \beta. \qquad (22)
\end{aligned}$$

Thus,

$$\sum_{e\in E^{ext}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))}\cdot\tau(e) \preccurlyeq E^{ext}_{single} + \sum_{\substack{f\in\text{image}(\tau)\\ f\in E^{ext}_{general}}} \beta\cdot f \preccurlyeq \beta\cdot F.$$

Plugging this last inequality into (21), we obtain

$$E^{ext} \preccurlyeq 3\sum_{i:|W_i|>1} \left(\sum_{e\in E^{ext}_i} \eta(e)\right) T_i + 3\beta\cdot F.$$

Applying (18) and then (16), we compute

$$E = E^{ext} + E^{int} \preccurlyeq 3 \sum_{i:|W_i|>1} T_i \left(\sum_{e \in E_i^{int}} \eta(e) + \sum_{e \in E_i^{ext}} \eta(e) \right) + 3\beta \cdot F \preccurlyeq 3\beta \cdot (T \cup F),$$

which by (15) implies the lemma. □

We now observe three sources of slack in Theorem 9.5, in decreasing order of importance. The first is the motivation for the construction of ultra-sparsifiers in the next section.

1. In the proof of Theorem 9.5, we assume in the worst case that the tree decomposition could result in each tree T_i being connected to $t-1$ other trees, for a total of $t(t-1)/2$ extra edges. Most of these edges seem barely necessary, as they could be included at a small fraction of their original weight. To see why, consider the crude estimate at the end of inequality (22). We upper bound the multiplier of one bridge edge f from T_i,

   $$\sum_{\substack{e \in E_i^{ext} \\ \tau(e)=f}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))},$$

   by the sum of the multipliers of all bridge edges from T_i,

   $$\sum_{e \in E_i^{ext}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))}.$$

   The extent to which this upper bound is loose is the factor by which we could decrease the weight of the edge f in the preconditioner.

   While we cannot accelerate our algorithms by decreasing the weights with which we include edges, we are able to use sparsifiers to trade many low-weight edges for a few edges of higher weight. This is how we reduce the number of edges we add to the spanning tree to $t \log^{O(1)} n$.

2. The number of edges added equals the number of pairs of trees that are connected. While we can easily obtain an upper bound on this quantity when the graph has bounded genus, it seems that we should also be able to bound this quantity when the graph has some nice geometrical structure.

3. The constant 12 in inequality (14) can be closer to 2 in practice. To see why, first note that the internal and external edges count quite differently: the external edges have three times as much impact. However, most of the edges will probably be internal. In fact, if one uses the algorithm of [ABN08] to construct the tree, then one can incorporate the augmentation into this process to minimize the number of external edges. Another factor of 2 can be saved by observing that the `decomposeTree`, as stated, counts the internal edges twice, but could be modified to count them once.

10 Ultra-Sparsifiers

We begin our construction of ultra-sparsifiers by building ultra-sparsifiers for the special case in which our graph has a distinguished vertex r and a low-stretch spanning tree T with the property that for every edge $e \in E - T$, the path in T connecting the endpoints of e goes through r. In this case, we will call r the *root* of the tree. All of the complications of ultra-sparsification will be handled in this construction. The general construction will follow simply by using tree splitters to choose the roots and decompose the input graph.

The algorithm `RootedUltraSparsify` begins by computing the same set of edges $\sigma(i,j)$, as was computed by `UltraSimple`. However, when `RootedUltraSparsify` puts one of these edges into the set F, it gives it a different weight: $\omega(i,j)$. For technical reasons, the set F is decomposed into subsets F^b according to the quantities $\phi(f)$, which will play a role in the analysis of `RootedUltraSparsify` analogous to the role played by $\eta(e)$ in the analysis of `UltraSimple`. Each set of edges F^b is sparsified, and the union of the edges of E that appear in the resulting sparsifiers are returned by the algorithm. The edges in F^b cannot necessarily be sparsified directly, as they might all have different endpoints. Instead, F^b is first projected to a graph H^b on vertex set $\{1,\ldots,h\}$. After a sparsifier H^b_s of H^b is computed, it is lifted back to the original graph to form E^b_s. Note that the graph E_s returned by `RootedUltraSparsify` is a subgraph of E, with the same edge weights.

We now prove that $F = \cup_{b=1}^{\lceil \log_2 \eta(E) \rceil} F^b$. Our proof will use the function η, which we recall was defined in (12) and which was used to define the map σ.

Lemma 10.1. *For ϕ as defined in (24), for every $f = \psi(i,j)\sigma(i,j) \in F$,*

$$1 \leq \psi(i,j) \leq \phi(f) \leq \eta(E). \tag{25}$$

Proof. Recall from the definitions of ϕ and ψ that

$$\phi(f) \geq \psi(i,j) = \frac{\sum_{e \in E: \rho(e) = \{W_i, W_j\}} \text{weight}(e)}{\text{weight}(\sigma(i,j))}.$$

By definition $\sigma(i,j)$ is an edge in E satisfying $\rho(\sigma(i,j)) = \{W_i, W_j\}$; so, the right-hand side of the last expression is at least 1.

To prove the upper bound on $\phi(f)$, first apply Proposition 9.4 to show that

$$\psi(i,j) = \frac{\sum_{e \in E: \rho(e) = \{W_i, W_j\}} \text{weight}(e)}{\text{weight}(\sigma(i,j))} \leq \frac{\sum_{e \in E: \rho(e) = \{W_i, W_j\}} \eta(e)}{\eta(\sigma(i,j))} \leq \eta(E),$$

as η is always at least 1. Similarly,

$$\begin{aligned}
\text{st}_T(f) = \frac{\omega(i,j)}{\text{weight}(\sigma(i,j))} \text{st}_T(\sigma(i,j)) &= \frac{\text{st}_T(\sigma(i,j))}{\text{weight}(\sigma(i,j))} \left(\sum_{e \in E: \rho(e) = \{W_i, W_j\}} \text{weight}(e) \right) \\
&\leq \frac{\eta(\sigma(i,j))}{\text{weight}(\sigma(i,j))} \left(\sum_{e \in E: \rho(e) = \{W_i, W_j\}} \text{weight}(e) \right) \leq \sum_{e \in E: \rho(e) = \{W_i, W_j\}} \eta(e) \leq \eta(E),
\end{aligned}$$

where the second-to-last inequality follows from Proposition 9.4. ∎

$E_s = \texttt{RootedUltraSparsify}(E, T, r, t, p)$

Condition: for all $e \in E$, $r \in T(e)$. The parameter t is a positive integer at most $\lceil \eta(E) \rceil$.

1. Compute $\mathrm{st}_T(e)$ and $\eta(e)$ for each edge $e \in E$, where η is as defined in (12).
2. If $t \geq |E|$, return $E_s = E$.
3. Set $(\{W_1, \ldots, W_h\}, \rho) = \texttt{decompose}(T, E, \eta, t)$.
4. Compute σ, as given by (13), everywhere it is defined.
5. For every (i, j) such that $\sigma(i, j)$ is defined, set

$$\omega(i,j) = \sum_{e \in E : \rho(e) = \{W_i, W_j\}} \mathrm{weight}(e) \quad \text{and} \quad \psi(i,j) = \omega(i,j)/\mathrm{weight}(\sigma(i,j)). \tag{23}$$

6. Set $F = \{\psi(i,j)\sigma(i,j) : \sigma(i,j) \text{ is defined}\}$.
7. For each $f = \psi(i,j)\sigma(i,j) \in F$, set

$$\phi(f) = \max(\psi(i,j), \mathrm{st}_T(f)). \tag{24}$$

8. For $b \in \{1, \ldots, \lceil \log_2 \eta(E) \rceil\}$:
   (a) Set $F^b = \begin{cases} \{f \in F : \phi(f) \in [1,2]\} & \text{if } b = 1 \\ \{f \in F : \phi(f) \in (2^{b-1}, 2^b]\} & \text{otherwise} \end{cases}$
   (b) Let H^b be the set of edges on vertex set $\{1, \ldots, h\}$ defined by

$$H^b = \left\{ \omega(i,j)\mathbf{(}i,j\mathbf{)} : \psi(i,j)\sigma(i,j) \in F^b \right\}.$$

   (c) Set $H^b_s = \texttt{Sparsify2}(H^b, p)$.
   (d) Set

$$E^b_s = \left\{ \sigma(i,j) : \exists w \text{ such that } w\mathbf{(}i,j\mathbf{)} \in H^b_s \right\}.$$

9. Set $E_s = \cup_b E^b_s$.

It will be convenient for us to extend the domain of ρ to F by setting $\rho(f) = \rho(e)$ where $e \in E$ has the same vertices as f. That is, when there exists $\gamma \in \mathbb{R}^+$ such that $f = \gamma e$. Define

$$\beta = 4\eta(E)/t.$$

Our analysis of `RootedUltraSparsify` will exploit the inequalities contained in the following two lemmas.

Lemma 10.2. *For every i for which $|W_i| > 1$,*

$$\sum_{f \in F : W_i \in \rho(f)} st_T(f) \leq \beta.$$

Proof. Consider any $f \in F$, and let $f = \psi(i,j)\sigma(i,j)$. Note that the weight of f is $\omega(i,j)$, and recall that $\mathrm{st}_T(f) \leq \eta(f)$. We first show that

$$\sum_{e : \tau(e) = \sigma(i,j)} \eta(e) \geq \eta(f).$$

By Proposition 9.4, and the definition of τ in (19)

$$\begin{aligned}
\sum_{e : \tau(e) = \sigma(i,j)} \eta(e) &\geq \frac{\eta(\sigma(i,j))}{\mathrm{weight}(\sigma(i,j))} \sum_{e : \tau(e) = \sigma(i,j)} \mathrm{weight}(e) \\
&= \frac{\eta(\sigma(i,j))}{\mathrm{weight}(\sigma(i,j))} \mathrm{weight}(f) \\
&= \max\left(\frac{\mathrm{weight}(f)}{\mathrm{weight}(\sigma(i,j))}, \frac{\mathrm{st}_T(\sigma(i,j))}{\mathrm{weight}(\sigma(i,j))} \mathrm{weight}(f) \right) \\
&= \max\left(\psi(i,j), \mathrm{st}_T(f)\right) \\
&= \max\left(\phi(f), \mathrm{st}_T(f)\right) \quad \text{(by (24))} \\
&\geq \max\left(1, \mathrm{st}_T(f)\right) \quad \text{(by (25))} \\
&= \eta(f).
\end{aligned}$$

We then have

$$\sum_{e \in E : W_i \in \rho(e)} \eta(e) \geq \sum_{f \in F : W_i \in \rho(f)} \eta(f).$$

The lemma now follows from the upper bound of $4\eta(E)/t$ imposed on the left-hand term by Theorem 9.3. □

Lemma 10.3. *For every i for which $|W_i| > 1$,*

$$\sum_{f \in F : W_i \in \rho(f)} \phi(f) \leq 2\beta. \tag{26}$$

Proof. For an edge $f \in F$, let $\psi(f)$ equal $\psi(i,j)$ where $f = \psi(i,j)\sigma(i,j)$. With this notation, we may compute

$$\begin{aligned}\sum_{f \in F : W_i \in \rho(f)} \phi(f) &\leq \sum_{f \in F : W_i \in \rho(f)} \mathrm{st}_T(f) + \sum_{f \in F : W_i \in \rho(f)} \psi(f) \\ &\leq \sum_{f \in F : W_i \in \rho(f)} \eta(f) + \sum_{f \in F : W_i \in \rho(f)} \psi(f) \\ &\leq \beta + \sum_{f \in F : W_i \in \rho(f)} \psi(f),\end{aligned}$$

by Lemma 10.2. We now bound the right-hand term as in the proof of inequality (22):

$$\sum_{f \in F : W_i \in \rho(f)} \psi(f) = \sum_{e \in E_i^{ext}} \frac{\mathrm{weight}(e)}{\mathrm{weight}(\tau(e))} \leq \sum_{e \in E_i^{ext}} \frac{\eta(e)}{\eta(\tau(e))} \leq \sum_{e \in E_i^{ext}} \eta(e) \leq \beta,$$

by our choice of β and Theorem 9.3. □

Lemma 10.4 (`RootedUltraSparsify`)**.** *Let T be a spanning tree on a vertex set V, and let E be a non-empty set of edges on V for which there exists an $r \in V$ be such that for all $e \in E$, $r \in T(v)$. For $p > 0$ and t a positive integer at most $\lceil \eta(E) \rceil$, let E_s be the graph returned by `RootedUltraSparsify`(E, T, r, t, p). The graph E_s is a subgraph of E, and with probability at least $1 - \lceil \log_2 \eta(E) \rceil \, p$,*

$$|E_s| \leq c_1 \log^{c_2}(n/p) \max(1, \lceil \log_2 \eta(E) \rceil) t, \tag{27}$$

and

$$E \preccurlyeq (3\beta + 126\beta \max(1, \log_2 \eta(E))) \cdot T + 120\beta \cdot E_s, \tag{28}$$

where $\beta = 4\eta(E)/t$.

Proof. We first dispense with the case in which the algorithm terminates at line 2. If $t \geq m$, then both (27) and (28) are trivially satisfied by setting $E_s = E$, as $\beta \geq 2$.

By Theorem 1.3 each graph H_s^b computed by `Sparsify2` is a $c_1 \log^{c_2}(n/p)$-sparsifier of H^b according to Definition 1.2 with probability at least $1 - p$. As there are at most $\lceil \log_2 \eta(E) \rceil$ such graphs H^b, this happens for all of these graphs with probability at least $1 - \lceil \log_2 \eta(E) \rceil \, p$. For the remainder of the proof, we will assume that each graph H_s^b is a $c_1 \log^{c_2}(n/p)$-sparsifier of H^b. Recalling that $h \leq t$, the bound on the number of edges in E_s is immediate.

Our proof of (28) will go through an analysis of intermediate graphs. As some of these could be multi-graphs, we will find it convenient to write them as sums of edges.

To define these intermediate graphs, let r_i be the vertex in W_i that is closest to r in T. As in Section 9, let T_i denote the edges of the subtree of T with vertex set W_i. We will view r_i as the root of tree T_i. Note that if $|W_i| = 1$, then $W_i = \{r_i\}$ and T_i is empty. As distinct sets W_i and W_j can overlap in at most one vertex, $\sum_i T_i \leq T$. We will exploit the fact that for each $e \in E$ with $\rho(e) = \{W_i, W_j\}$, the path $T(e)$ contains both r_i and r_j, which follows from the condition $r \in T(e)$.

We now define the edge set D^b, which is a projection of H^b to the vertex set $r_1, \ldots, r_h$, and D^b_s, which is an analogous projection of the sparsifier H^b_s. We set

$$D^b = \sum_{(i,j):\psi(i,j)\sigma(i,j)\in F^b} \omega(i,j)\big(r_i, r_j\big)$$

and

$$D^b_s = \sum_{w\big(i,j\big)\in H^b_s} w\big(r_i, r_j\big).$$

As the sets W_i and W_j are allowed to overlap slightly, it could be the case that some $r_i = r_j$ for $i \neq j$. In this case, D^b would not be isomorphic to H^b.

Set

$$F^b_s = \left\{\gamma\psi(i,j)\sigma(i,j) : \exists\gamma \text{ and } (i,j) \text{ so that } \gamma\omega(i,j)\big(i,j\big) \in H^b_s\right\}.$$

The edge set H^b can be viewed as a projection of the edge set F^b to the vertex set $\{1, \ldots, h\}$, and the edge set F^b_s can be viewed as a lift of H^b_s back into a reweighted subgraph of F^b.

We will prove the following inequalities

$$E \preccurlyeq 3\beta \cdot T + 3F \tag{29}$$

$$F^b \preccurlyeq 2\beta \cdot T + 2D^b \tag{30}$$

$$D^b \preccurlyeq (5/4)D^b_s \tag{31}$$

$$D^b_s \preccurlyeq 16\beta \cdot T + 2F^b_s \tag{32}$$

$$F^b_s \preccurlyeq 8\beta \cdot E^b_s \tag{33}$$

Inequality (28) in the statement of the lemma follows from these inequalities and $F = \sum_b F^b$.

To prove inequality (29), we exploit the proof of Theorem 9.5. The edges F constructed in `RootedUltraSparsify` are the same as those chosen by `UltraSimple`, except that they are reweighted by the function ψ. If we follow the proof of inequality (14) in Theorem 9.5, but neglect to apply inequality (22), we obtain

$$E \preccurlyeq 3\beta \cdot T + 3 \sum_{e\in E^{ext}} \frac{\text{weight}(e)}{\text{weight}(\tau(e))} \cdot \tau(e) = 3\beta \cdot T + 3F.$$

To prove inequality (30), consider any edge $w\big(u, v\big) = f \in F^b$. Assume $\rho(f) = \{W_i, W_j\}$, $u \in W_i$ and $v \in W_j$. We will now show that

$$f \preccurlyeq 2\text{st}_T(f)(T_i + T_j) + 2w\big(r_i, r_j\big). \tag{34}$$

As the path from u to v in T contains both r_i and r_j,

$$\text{resistance}(T(u, r_i)) + \text{resistance}(T(r_j, v)) \leq \text{resistance}(T(u, v)) = \text{st}_T(f)/w.$$

Thus, the resistance of the path

$$2\text{st}_T(f)T(u, r_i) + 2w\big(r_i, r_j\big) + 2\text{st}_T(f)T(r_j, v)$$

is at most $1/w$, and so Lemma 8.1 implies that

$$f \preccurlyeq 2\mathrm{st}_T(f)T(u,r_i) + 2w\big(r_i,r_j\big) + 2\mathrm{st}_T(f)T(r_j,v),$$

which in turn implies (34). Summing (34) over all $f \in F^b$ yields

$$\begin{aligned}
F^b &\preccurlyeq 2\sum_i \left(\sum_{f\in F: W_i\in\rho(f)} \mathrm{st}_T(f)\right) T_i + 2D^b \\
F^b &\preccurlyeq 2\sum_{i:|W_i|>1} \left(\sum_{f\in F: W_i\in\rho(f)} \mathrm{st}_T(f)\right) T_i + 2D^b && \text{as } T_i \text{ is empty when } |W_i| = 1 \\
&\preccurlyeq 2\sum_i \beta\cdot T_i + 2D^b, && \text{by Lemma 10.2} \\
&\preccurlyeq 2\beta\cdot T + 2D^b.
\end{aligned}$$

We now prove inequality (32), as it uses similar techniques. Let $f_s = w\big(u,v\big) \in F_s^b$. Then, there exist γ and (i,j) so that $\gamma\omega(i,j)\big(i,j\big) \in H_s^b$, $u \in W_i$, and $v \in W_j$. Set $\gamma(f_s)$ to be this multiplier γ. By part (c) of Definition 1.2, we must have $\omega(i,j)\big(i,j\big) \in H^b$ and $\psi(i,j)\sigma(i,j) \in F^b$. Let $f = \psi(i,j)\sigma(i,j)$. Note that $f_s = \gamma(f_s)f$. The sum of the resistances of the paths from r_i to u in T_i and from v to r_j in T_j is

$$\text{resistance}(T(r_i,u)) + \text{resistance}(T(v,r_j)) \le \text{resistance}(T(u,v)) = \mathrm{st}_T(f)/\omega(i,j),$$

as $\text{weight}(f) = \omega(i,j)$. Thus, the resistance of the path

$$2\mathrm{st}_T(f)T(r_i,u) + 2f + 2\mathrm{st}_T(f)T(v,r_j)$$

is at most $1/\omega(i,j)$, and so Lemma 8.1 implies that

$$\omega(i,j)\big(r_i,r_j\big) \preccurlyeq 2\mathrm{st}_T(f)(T_i+T_j) + 2f,$$

and

$$\begin{aligned}
\gamma(f_s)\omega(i,j)\big(r_i,r_j\big) &\preccurlyeq 2\gamma(f_s)\mathrm{st}_T(f)(T_i+T_j) + 2f_s \\
&\preccurlyeq 2\gamma(f_s)\phi(f)(T_i+T_j) + 2f_s && \text{(by (24))} \\
&\preccurlyeq 2^{b+1}\gamma(f_s)(T_i+T_j) + 2f_s && \text{(by } f\in F^b\text{)}.
\end{aligned}$$

Summing this inequality over all $f_s \in F_s^b$, we obtain

$$D_s^b \preccurlyeq \sum_i \left(2^{b+1}\sum_{f_s\in F_s^b: W_i\in\rho(f_s)} \gamma(f_s)\right) T_i + 2F_s^b.$$

For all i such that $|W_i| > 1$,

$$\begin{aligned}
\sum_{f_s\in F_s^b: W_i\in\rho(f_s)} \gamma(f_s) &\le 2\left|\left\{f\in F^b : W_i\in\rho(f)\right\}\right| && \text{(part } (d) \text{ of Definition 1.2)} \\
&\le 2\sum_{f\in F^b: W_i\in\rho(f)} \phi(f)/2^{b-1} \\
&\le 4\beta/2^{b-1} && \text{(by Lemma 10.3)} \\
&= \beta/2^{b-3}. && (35)
\end{aligned}$$

So,

$$D_s^b \preccurlyeq \sum_i 16\beta \cdot T_i + 2F_s^b \preccurlyeq 16\beta \cdot T + 2F_s^b.$$

To prove inequality (33), let f_s be any edge in F_s, let f be the edge in F such that $f_s = \gamma(f_s)f$, and let $\sigma(i,j)$ be the edge such that $f_s = \gamma(f_s)\psi(i,j)\sigma(i,j)$. It suffices to show that

$$\text{weight}(f_s) \leq 8\beta \text{ weight}(\sigma(i,j)). \tag{36}$$

Set b so that $f \in F^b$. By (35),

$$\gamma(f_s) \leq \beta/2^{b-3} \leq 8\beta/\phi(f) = 8\beta/\max(\psi(i,j), \text{st}_T(f)) \leq 8\beta/\psi(i,j).$$

As $\text{weight}(f_s) = \gamma(f_s)\psi(i,j)\text{weight}(\sigma(i,j))$, inequality (36) follows.

It remains to prove inequality (31). The only reason this inequality is not immediate from part (a) of Definition 1.2 is that we may have $r_i = r_j$ for some $i \neq j$. Let $R = \{r_1, \ldots, r_h\}$ and $S = \{1, \ldots, h\}$, Define the map $\pi : \mathbb{R}^R \to \mathbb{R}^S$ by $\pi(x)_i = x_{r_i}$. We then have for all $x \in \mathbb{R}^R$

$$x^T L_{D^b} x = \pi(x)^T L_{H^b} \pi(x) \quad \text{and} \quad x^T L_{D_s^b} x = \pi(x)^T L_{H_s^b} \pi(x);$$

so,

$$x^T L_{D^b} x = \pi(x)^T L_{H^b} \pi(x) \leq (5/4)\pi(x)^T L_{H_s^b} \pi(x) = (5/4) x^T L_{D_s^b} x.$$

□

The algorithm `UltraSparsify` will construct a low-stretch spanning tree T of a graph, choose a root vertex r, apply `RootedUltraSparsify` to sparsify all edges whose path in T contains r, and then work recursively on the trees obtained by removing the root vertex from T. The root vertex will be chosen to be a tree *splitter*, where we recall that a vertex r is a *splitter* of a tree T if the trees $T^1, \ldots, T^q$ obtained by removing r each have at most two-thirds as many vertices as T. It is well-known that a tree splitter can be found in linear time. By making the root a splitter of the tree, we bound the depth of the recursion. This is both critical for bounding the running time of the algorithm and for proving a bound on the quality of the approximation it returns. For each edge e such that $r \notin T(e)$, $T(e)$ is entirely contained in one of $T^1, \ldots, T^q$. Such edges are sparsified recursively.

$U =$ `UltraSparsify`$(G = (V, E), k)$
Condition: G is connected.

1. $T =$ `LowStretch`(E).
2. Set $t = 517 \cdot \max(1, \log_2 \eta(E)) \cdot \left\lceil \log_{3/2} n \right\rceil \eta(E)/k$ and $p = \left(2 \lceil \log \eta(E) \rceil n^2\right)^{-1}$.
3. If $t \geq \eta(E)$ then set $A = E - T$; otherwise, set $A =$ `TreeUltraSparsify`$(E - T, t, T, p)$.
4. $U = T \cup A$.

$A =$ `TreeUltraSparsify`(E', t', T', p)

1. If $E' = \emptyset$, return $A = \emptyset$.
2. Compute a splitter r of T'.
3. Set $E^r = \{\text{edges } e \in E' \text{ such that } r \in T'(e)\}$ and $t_r = \lceil t'\eta(E^r)/\eta(E') \rceil$.
4. If $t_r > 1$, set $A^r =$ `RootedUltraSparsify`(E^r, T', r, t_r, p); otherwise, set $A^r = \emptyset$.
5. Set $T^1, \ldots, T^q$ to be the trees obtained by removing r from T'. Set $V^1, \ldots, V^q$ to be the vertex sets of these trees, and set $E^1, \ldots, E^q$ so that $E^i = \left\{(u,v) \in E' : \{u,v\} \subseteq V^i\right\}$.
6. For $i = 1, \ldots, q$, set

$$A = A^r \cup \texttt{TreeUltraSparsify}(E^i, t'\eta(E^i)/\eta(E'), T^i, p).$$

Theorem 10.5 (Ultra-Sparsification)**.** *On input a weighted, connected n-vertex graph $G = (V, E)$ and $k \geq 1$, `UltraSparsify`(E, k) returns a set of edges $U = T \cup A \subseteq E$ such that T is a spanning tree of G, $U \subseteq E$, and with probability at least $1 - 1/2n$,*

$$U \preccurlyeq E \preccurlyeq kU, \tag{37}$$

and

$$|A| \leq O\left(\frac{m}{k} \log^{c_2+5} n\right), \tag{38}$$

where $m = |E|$. Furthermore, `UltraSparsify` runs in expected time $m \log^{O(1)} n$.

We remark that this theorem is very loose when $m/k \geq n$. In this case, the calls made to `decompose` by `RootedUltraSparsify` could have $t \geq n$, in which case `decompose` will just return singleton sets, and the output of `RootedUltraSparsify` will essentially just be the output of `Sparsify2` on E^r. In this case, the upper bound in (38) can be very loose.

Proof. We first dispense with the case $t \geq \eta(E)$. In this case, `UltraSparsify` simply returns the graph E, so (37) is trivially satisfied. The inequality $t \geq \eta(E)$ implies $k \leq O(\log^2 n)$, so (38) is trivially satisfied as well.

At the end of the proof, we will use the inequality $t < \eta(E)$. It will be useful to observe that every time `TreeUltraSparsify` is invoked,

$$t' = t\eta(E')/\eta(E).$$

To apply the analysis of `RootedUltraSparsify`, we must have

$$t^r \leq \lceil \eta(E^r) \rceil .$$

This follows from

$$t^r = \left\lceil t'\eta(E^r)/\eta(E') \right\rceil = \lceil t\eta(E^r)/\eta(E) \rceil \leq \lceil \eta(E^r) \rceil ,$$

as `TreeUltraSparsify` is only called if $t < \eta(E)$.

Each vertex of V can be a root in a call to `RootedUltraSparsify` at most once, so this subroutine is called at most n times during the execution of `UltraSparsify`. Thus by Lemma 10.4, with probability at least

$$1 - n \lceil \log_2 \eta(E) \rceil \, p = 1 - 1/2n,$$

every graph E_s returned by a call to `RootedUltraSparsify` satisfies (27) and (28). Accordingly, we will assume both of these conditions hold for the rest of our analysis.

We now prove the upper bound on the number of edges in A. During the execution of `UltraSparsify`, many vertices become the root of some tree. For those vertices v that do not, set $t_v = 0$. By (27),

$$|A| = \sum_{r \in V : t_r > 1} |A^r| \leq c_1 \log^{c_2}(n/p) \max(1, \lceil \log_2 \eta(E) \rceil) \sum_{r \in V : t_r > 1} t_r. \tag{39}$$

As $\lceil z \rceil \leq 2z$ for $z \geq 1$ and $E^{r_1} \cap E^{r_2} = \emptyset$ for each $r_1 \neq r_2$,

$$\sum_{r \in V : t_r > 1} t_r = \sum_{r \in V : t_r > 1} \left\lceil \frac{\eta(E^r)}{\eta(E)} t \right\rceil \leq \sum_{r \in V : t_r > 1} \frac{2\eta(E^r)}{\eta(E)} t \leq 2t.$$

Thus,

$$\begin{aligned}
(39) &\leq 2c_1 \log^{c_2}(n/p) \lceil \log_2 \eta(E) \rceil \, t \\
&\leq 2c_1 \log^{c_2}(n/p) \lceil \log_2 \eta(E) \rceil \, 517 \cdot \log_2 \eta(E) \cdot \left\lceil \log_{3/2} n \right\rceil \eta(E)/k \\
&\leq O\left(\frac{m}{k} \log^{c_2+5} n \right),
\end{aligned}$$

where the last inequality uses $\eta(E) = O(m \log n \log^2 n) = O(m \log^2 n)$ from Theorem 9.1 and $\log m = O(\log n)$.

We now establish (37). For every vertex r that is ever selected as a tree splitter in line 2 of `TreeUltraSparsify`, let T^r be the tree T' of which r is a splitter, and let E^r denote the set of edges and t_r be the parameter set in line 3. Observe that $\cup_r E^r = E - T$. Let

$$\beta_r = 4\eta(E^r)/t_r,$$

and note this is the parameter used in the analysis of `RootedUltraSparsify` in Lemma 10.4. If $t_r > 1$, let A^r be the set of edges returned by the call to `RootedUltraSparsify`. By Lemma 10.4, `RootedUltraSparsify` returns a set of edges A^r satisfying

$$E^r \preccurlyeq (3\beta_r + 126\beta_r \max(1, \log_2 \eta(E^r))) \cdot T^r + 120\beta_r \cdot A^r. \tag{40}$$

On the other hand, if $t_r = 1$ and so $A^r = \emptyset$, then $\beta_r = 4\eta(E^r)$. We know that (40) is satisfied in this case because $E^r \preccurlyeq \eta(E^r) T^r$ (by (10)). If $t_r = 0$, then $E^r = \emptyset$ and (40) is trivially satisfied. As $t_r = \lceil t\eta(E^r)/\eta(E) \rceil$,

$$\beta_r \leq 4\eta(E)/t.$$

We conclude

$$E^r \preccurlyeq 129\beta_r \max(1, \log_2 \eta(E^r)) \cdot T^r + 120\beta_r \cdot A^r \preccurlyeq 516(\eta(E)/t) \max(1, \log_2 \eta(E^r)) T^r + 120(\eta(E)/t) A^r.$$

Adding T, summing over all r, and remembering $\eta(E^r) \leq \eta(E)$, we obtain

$$T + (E - T) \preccurlyeq T + 516(\eta(E)/t) \max(1, \log_2 \eta(E)) \sum_r T^r + 120(\eta(E)/t)A.$$

As r is always chosen to be a splitter of the tree input to `TreeUltraSparsify`, the depth of the recursion is at most $\left\lceil \log_{3/2} n \right\rceil$. Thus, no edge of T appears more than $\left\lceil \log_{3/2} n \right\rceil$ times in the sum $\sum_r T^r$, and we may conclude

$$\begin{aligned} T + (E - T) &\preccurlyeq T + 516(\eta(E)/t) \max(1, \log_2 \eta(E)) \left\lceil \log_{3/2} n \right\rceil T + 120(\eta(E)/t)A \\ &\preccurlyeq 517(\eta(E)/t) \max(1, \log_2 \eta(E)) \left\lceil \log_{3/2} n \right\rceil T + 120(\eta(E)/t)A \\ &\preccurlyeq k(T + A) \\ &= kU, \end{aligned}$$

where the second inequality follows from $t \leq \eta(E)$, and the third inequality follows from the value chosen for t in line 2 of `UltraSparsify`.

To bound the expected running time of `UltraSparsify`, first observe that the call to `LowStretch` takes time $O(m \log^2 n)$. Then, note that the routine `TreeUltraSparsify` is recursive, the recursion has depth at most $O(\log n)$, and all the graphs being processed by `TreeUltraSparsify` at any level of the recursion are disjoint. The running time of `TreeUltraSparsify` is dominated by the calls made to `Sparsify2` inside `RootedUltraSparsify`. Each of these takes nearly-linear expected time, so the overall expected running time of `TreeUltraSparsify` is $O(m \log^{O(1)} n)$. □

References

[ABN08] I. Abraham, Y. Bartal, and O. Neiman. Nearly tight low stretch spanning trees. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 781–790, Oct. 2008.

[AKPW95] Noga Alon, Richard M. Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, February 1995.

[Axe85] O. Axelsson. A survey of preconditioned iterative methods for linear systems of algebraic equations. *BIT Numerical Mathematics*, 25(1):165–187, March 1985.

[BBC+94] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[BCHT04] Erik G. Boman, Doron Chen, Bruce Hendrickson, and Sivan Toledo. Maximum-weight-basis preconditioners. *Numerical linear algebra with applications*, 11(8–9):695–721, October/November 2004.

[BGH+06] M. Bern, J. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo. Support-graph preconditioners. *SIAM J. Matrix Anal. & Appl*, 27(4):930–951, 2006.

[BH01] Erik Boman and B. Hendrickson. On spanning tree preconditioners. Manuscript, Sandia National Lab., 2001.

[BH03a] Mario Bebendorf and Wolfgang Hackbusch. Existence of H-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. *Numerische Mathematik*, 95(1):1–28, July 2003.

[BH03b] Erik G. Boman and Bruce Hendrickson. Support theory for preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 25(3):694–717, 2003.

[BHM01] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial, 2nd Edition*. SIAM, 2001.

[BHV04] Erik G. Boman, Bruce Hendrickson, and Stephen A. Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *CoRR*, cs.NA/0407022, 2004.

[BMN04] M. Belkin, I. Matveeva, and P. Niyogi. Regularization and semi-supervised learning on large graphs. *Proc. 17th Conf. on Learning Theory*, pages 624–638, 2004.

[BSS09] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. In *Proceedings of the 41st Annual ACM Symposium on Theory of computing*, pages 255–262, 2009.

[CW82] D. Coppersmith and S. Winograd. On the asymptotic complexity of matrix multiplication. *SIAM Journal on Computing*, 11(3):472–492, August 1982.

[DER86] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford Science Publications, 1986.

[DS07] Samuel I. Daitch and Daniel A. Spielman. Support-graph preconditioners for 2-dimensional trusses. *CoRR*, abs/cs/0703119, 2007.

[DS08] Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 451–460, 2008.

[EEST08] Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 32(2):608–628, 2008.

[EGB05] Ojas Parekh Sivan Toledo Erik G. Boman, Doron Chen. On factor width and symmetric h-matrices. *Linear Algebra and its Applications*, 405(1):239–248, August 2005.

[FG04] A. Frangioni and C. Gentile. New preconditioners for KKT systems of network flow problems. *SIAM Journal on Optimization*, 14(3):894–913, 2004.

[Fie73] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[Geo73] J. A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10:345–363, 1973.

[GMZ95] Keith D. Gremban, Gary L. Miller, and Marco Zagha. Performance evaluation of a new parallel preconditioner. In *Proceedings of the 9th International Symposium on Parallel Processing*, pages 65–69. IEEE Computer Society, 1995.

[GO88] G. H. Golub and M. Overton. The convergence of inexact Chebychev and Richardson iterative methods for solving linear systems. *Numerische Mathematik*, 53:571–594, 1988.

[Gre96] Keith Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, Carnegie Mellon University, CMU-CS-96-123, 1996.

[GT87] John R. Gilbert and Robert Endre Tarjan. The analysis of a nested dissection algorithm. *Numerische Mathematik*, 50(4):377–404, February 1987.

[Har72] Frank Harary. *Graph Theory*. Addison-Wesley, 1972.

[Jos97] Anil Joshi. *Topics in Optimization and Sparse Linear Systems*. PhD thesis, UIUC, 1997.

[KM07] Ioannis Koutis and Gary L. Miller. A linear work, $o(n^{1/6})$ time, parallel algorithm for solving planar Laplacians. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1002–1011, 2007.

[LRT79] Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, April 1979.

[Mih89] Milena Mihail. Conductance and convergence of Markov chains—A combinatorial treatment of expanders. In *30th Annual IEEE Symposium on Foundations of Computer Science*, pages 526–531, 1989.

[MMP+05] Bruce M. Maggs, Gary L. Miller, Ojas Parekh, R. Ravi, and Shan Leung Maverick Woo. Finding effective support-tree preconditioners. In *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 176–185, 2005.

[Rei98] John Reif. Efficient approximate solution of sparse linear systems. *Computers and Mathematics with Applications*, 36(9):37–58, 1998.

[SS08] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the 40th annual ACM Symposium on Theory of Computing*, pages 563–568, 2008.

[SST06] A. Sankar, D. A. Spielman, and S.-H. Teng. Smoothed analysis of the condition numbers and growth factors of matrices. *SIAM Journal on Matrix Analysis and Applications*, 28(2):446–476, 2006.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM Symposium on Theory of Computing*, pages 81–90, 2004. Full version available at `http://arxiv.org/abs/cs.DS/0310051`.

[ST08a] Gil Shklarski and Sivan Toledo. Rigidity in finite-element matrices: Sufficient conditions for the rigidity of structures and substructures. *SIAM Journal on Matrix Analysis and Applications*, 30(1):7–40, 2008.

[ST08b] Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. *CoRR*, abs/0809.3232, 2008. Available at `http://arxiv.org/abs/0809.3232`.

[ST08c] Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *CoRR*, abs/0808.4134, 2008. Available at `http://arxiv.org/abs/0808.4134`.

[Str86] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[SW09] Daniel A. Spielman and Jaeoh Woo. A note on preconditioning by low-stretch spanning trees. *CoRR*, abs/0903.2816, 2009. Available at `http://arxiv.org/abs/0903.2816`.

[TB97] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, Philadelphia, PA, 1997.

[Vai90] Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript UIUC 1990. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991, Minneapolis., 1990.

[ZBL+03] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Adv. in Neural Inf. Proc. Sys. 16*, pages 321–328, 2003.

[ZGL03] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proc. 20th Int. Conf. on Mach. Learn.*, 2003.

A Gremban's reduction

Gremban [Gre96] (see also [MMP+05]) provides the following method for handling positive off-diagonal entries. If A is a SDD_0-matrix, then Gremban decomposes A into $D + A_n + A_p$, where

D is the diagonal of A, A_n is the matrix containing all the negative off-diagonal entries of A, and A_p contains all the positive off-diagonals. Gremban then considers the linear system

$$\widehat{A}\begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix} = \hat{\boldsymbol{b}}, \qquad \text{where} \qquad \widehat{A} = \begin{bmatrix} D + A_n & -A_p \\ -A_p & D + A_n \end{bmatrix} \quad \text{and} \quad \hat{\boldsymbol{b}} = \begin{pmatrix} \boldsymbol{b} \\ -\boldsymbol{b} \end{pmatrix},$$

and observes that $\boldsymbol{x} = (\boldsymbol{x}_1 - \boldsymbol{x}_2)/2$ will be the solution to $A\boldsymbol{x} = \boldsymbol{b}$, if a solution exists. Moreover, approximate solutions of Gremban's system yield approximate solutions of the original:

$$\left\| \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix} - \widehat{A}^\dagger \hat{\boldsymbol{b}} \right\| \leq \epsilon \left\| \widehat{A}^\dagger \hat{\boldsymbol{b}} \right\| \qquad \text{implies} \qquad \left\| \boldsymbol{x} - A^\dagger \boldsymbol{b} \right\| \leq \epsilon \left\| A^\dagger \boldsymbol{b} \right\|,$$

where again $\boldsymbol{x} = (\boldsymbol{x}_1 - \boldsymbol{x}_2)/2$. Thus we may reduce the problem of solving a linear system in a SDD_0-matrix into that of solving a linear system in a SDDM_0-matrix that is at most twice as large and has at most twice as many non-zero entries.

B Computing the stretch

We now show that given a weighted graph $G = (V, E)$ and a spanning tree T of G, we can compute $\text{st}_T(e)$ for every edge $e \in E$ in $O((m+n)\log n)$ time, where $m = |E|$ and $n = |V|$.

For each pair of vertices $u, v \in V$, let resistance(u, v) be the resistance of $T(u, v)$, the path in T connecting u and v. We first observe that for an arbitrary $r \in V$, we can compute resistance(v, r) for all $v \in V$ in $O(n)$ time by a top-down traversal on the rooted tree obtained from T with root r. Using this information, we can compute the stretch of all edges in $E_r = \{\text{edges } e \in E \text{ such that } r \in T(e)\}$ in time $O(|E_r|)$. We can then use tree splitters in the same manner as in `TreeUltraSparsify` to compute the stretch of all edges in E in $O((m+n)\log n)$ time.

C Decomposing Trees

The pseudo-code for for `decompose` appears on the next page. The algorithm performs a depth-first traversal of the tree, greedily forming sets W_i once they are attached to a sufficient number of edges of E. While these sets are being created, the edges they are responsible for are stored in F_{sub}, and the sum of the value of η on these edges is stored in w_{sub}. When a set W_i is formed, the edges e for which $\rho(e) = W_i$ are set to some combination of F_{sub} and F_v.

We assume that some vertex r has been chosen to be the root of the tree. This choice is used to determine which nodes in the tree are children of each other.

Proof *of Theorem 9.3.* As algorithm `decompose` traverses the tree T once and visits each edge in E once, it runs in linear time.

In our proof, we will say that an edge e is assigned to a set W_j if $W_j \in \rho(e)$. To prove part (a) of the theorem, we use the following observations: If W_j is formed in step 3.c.ii or step 6.b, then the sum of η over edges assigned to W_j is at least ϕ, and if W_j is formed in step 7.b, then the sum of η of edges incident to W_j and W_{j+1} (which is a singleton) is at least 2ϕ. Finally, if a set W_h is formed in line 5.b of `decompose`, then the sum of η over edges assigned to W_h is

$(\{W_1, \ldots, W_h\}, \rho) = \texttt{decompose}(T, E, \eta, t)$

Comment: h, ρ, and the W_i's are treated as global variables.

1. Set $h = 0$.
2. For all $e \in E$, set $\rho(e) = \emptyset$.
3. Set $\phi = 2\sum_{e \in E} \eta(e)/t$.
4. $(F, w, U) = \texttt{sub}(r)$.
5. If $U \neq \emptyset$,
   (a) $h = h + 1$.
   (b) $W_h = U$.
   (c) For all $e \in F$, set $\rho(e) = \rho(e) \cup \{W_h\}$.

$(F, w, U) = \texttt{sub}(v)$

1. Let $v_1, \ldots, v_s$ be the children of v.
2. Set $w_{sub} = 0$, $F_{sub} = \emptyset$ and $U_{sub} = \emptyset$.
3. For $i = 1, \ldots, s$
   (a) $(F_i, w_i, U_i) = \texttt{sub}(v_i)$.
   (b) $w_{sub} = w_{sub} + w_i$, $F_{sub} = F_{sub} \cup F_i$, $U_{sub} = U_{sub} \cup U_i$.
   (c) If $w_{sub} \geq \phi$,
      i. $h = h + 1$.
      ii. Set $W_h = U_{sub} \cup \{v\}$.
      iii. For all $e \in F_{sub}$, set $\rho(e) = \rho(e) \cup \{W_h\}$.
      iv. Set $w_{sub} = 0$, $F_{sub} = \emptyset$ and $U_{sub} = \emptyset$.
4. Set $F_v = \{(u, v) \in E\}$, the edges attached to v.
5. Set $w_v = \sum_{e \in F_v} \eta(e)$.
6. If $\phi \leq w_v + w_{sub} \leq 2\phi$,
   (a) $h = h + 1$.
   (b) Set $W_h = U_{sub} \cup \{v\}$.
   (c) For all $e \in F_{sub} \cup F_v$, set $\rho(e) = \rho(e) \cup \{W_h\}$.
   (d) Return $(\emptyset, 0, \emptyset)$.
7. If $w_v + w_{sub} > 2\phi$,
   (a) $h = h + 1$.
   (b) Set $W_h = U_{sub}$.
   (c) For all $e \in F_{sub}$, set $\rho(e) = \rho(e) \cup \{W_h\}$.
   (d) $h = h + 1$.
   (e) Set $W_h = \{v\}$.
   (f) For all $e \in F_v$, set $\rho(e) = \rho(e) \cup \{W_h\}$.
   (g) Return $(\emptyset, 0, \emptyset)$.
8. Return $(F_{sub} \cup F_v, w_{sub} + w_v, U_{sub} \cup \{v\})$

greater than zero. But, at most one set is formed this way. As each edge is assigned to at most two sets in $W_1, \ldots, W_h$, we may conclude

$$2\sum_{e \in E} \eta(e) > (h-1)\phi,$$

which implies $t > h - 1$. As both t and h are integers, this implies $t \geq h$.

We now prove part (b). First, observe that steps 6 and 7 guarantee that when a call to $\texttt{sub}(v)$ returns a triple (F, w, U),

$$w = \sum_{e \in U} \eta(e) < \phi.$$

Thus, when a set W_h is formed in step 3.c.ii, we know that the sum of η over edges assigned to W_h equals w_{sub} and is at most 2ϕ. Similarly, we may reason that $w_{sub} < \phi$ at step 4. If a set W_h is formed in step 6.b, the sum of η over edges associated with W_h is $w_v + w_{sub}$, and must be at most 2ϕ. If a set W_h is formed in step 7.b, the sum of η over edges associated with W_h is w_{sub}, which we established is at most ϕ. As the set formed in step 7.e is a singleton, we do not need to bound the sum of η over its associated edges. □

Lemma C.1. *Suppose $G = (V, E)$ is a planar graph, π is a planar embedding of G, T is a spanning tree of G, and $t > 1$ is an integer. Let $(\{W_1, \ldots, W_h\}, \rho) = \texttt{decompose}(T, E, \eta, t)$ with the assumption that in Step 1 of $\texttt{sub}$, the children $v_1, \ldots, v_s$ of v always appear in clock-wise order according to π. Then the graph $G_{\{W_1,\ldots,W_h\}} = (\{1, \ldots, h\}, \{(i,j) : \exists\, e \in E, \rho(e) = \{W_i, W_j\}\})$ is planar.*

Proof. Recall that the contraction of an edge $e = (u, v)$ in a planar graph $G = (V, E)$ defines a new graph $(V - \{u\}, E \cup \{(x, v) : (x, u) \in E\} - \{(x, u) \in E\})$. Also recall that edge deletions and edge contractions preserve planarity.

We first prove the lemma in the special case in which the sets $W_1, \ldots, W_h$ are disjoint. For each j, let T_j be the graph induced on T by W_j. As each T_j is connected, $G_{\{W_1,\ldots,W_h\}}$ is a subgraph of the graph obtained by contracting all the edges in each subgraph T_j. Thus in this special case $G_{\{W_1,\ldots,W_h\}}$ is planar.

We now analyze the general case, recalling that the sets $W_1, \ldots, W_h$ can overlap. However, the only way sets W_j and W_k with $j < k$ can overlap is if the set W_j was formed at step 3.c.ii, and the vertex v becomes part of W_k after it is returned by a call to $\texttt{sub}$. In this situation, no edge is assigned to W_j for having v as an end-point. That is, the only edges of form (x, v) that can be assigned to W_j must have $x \in W_j$. So, these edges will not appear in $G_{\{W_1,\ldots,W_h\}}$.

Accordingly, for each j we define

$$X_j = \begin{cases} W_j - v & \text{if } W_j \text{ was formed at step 3.c.ii, and} \\ W_j & \text{otherwise.} \end{cases}$$

We have shown that $G_{\{W_1,\ldots,W_h\}} = G_{\{X_1,\ldots,X_h\}}$. Moreover, the sets $X_1, \ldots, X_h$ are disjoint. Our proof would now be finished, if only each subgraph of G induced by a set X_j were connected. While this is not necessarily the case, we can make it the case by adding edges to E.

The only way the subgraph of G induced on a set X_j can fail to be connected is if W_j is formed at line 3.c.ii from the union of v with a collection sets U_i for $i_0 \leq i \leq i_1$ returned by

recursive calls to `sub`. Now, consider what happens if we add edges of the form (v_i, v_{i+1}) to the graph for $i_0 \leq i < i_1$, whenever they are not already present. As the vertices $v_{i_0}, \ldots, v_{i_1}$ appear in clock-wise order around v, the addition of these edges preserves the planarity of the graph. Moreover, their addition makes the induced subgraphs on each set X_j connected, so we may conclude that $G_{\{X_1,\ldots,X_h\}}$ is in fact planar. □

D The Pseudo-Inverse of a Factored Symmetric Matrix

We recall that $B^\dagger$ is the pseudo-inverse of B if and only if it satisfies

$$BB^\dagger B = B \tag{41}$$
$$B^\dagger BB^\dagger = B^\dagger \tag{42}$$
$$(BB^\dagger)^T = BB^\dagger \tag{43}$$
$$(B^\dagger B)^T = B^\dagger B. \tag{44}$$

We now prove that if $B = XCX^T$, where X is a non-singular matrix and C is symmetric, then

$$B^\dagger = \Pi X^{-T} C^\dagger X^{-1} \Pi,$$

where Π is the projection onto the span of B. We will prove that by showing that $\Pi X^{-T} C^\dagger X^{-1} \Pi$ satisfies axioms (41–44). Recall that $\Pi = BB^\dagger = B^\dagger B$ and that $\Pi B = B$.

To verify (41), we compute

$$\begin{aligned} B(\Pi X^{-T} C^\dagger X^{-1} \Pi)B &= BX^{-T} C^\dagger X^{-1} B \\ &= (XCX^T) X^{-T} C^\dagger X^{-1} (XCX^T) \\ &= XCC^\dagger CX^T \\ &= XCX^T \\ &= B. \end{aligned}$$

To verify (42), we compute

$$\begin{aligned} (\Pi X^{-T} C^\dagger X^{-1} \Pi)B(\Pi X^{-T} C^\dagger X^{-1} \Pi) &= \Pi X^{-T} C^\dagger X^{-1} BX^{-T} C^\dagger X^{-1} \Pi \\ &= \Pi X^{-T} C^\dagger X^{-1} XCX^T X^{-T} C^\dagger X^{-1} \Pi \\ &= \Pi X^{-T} C^\dagger CC^\dagger X^{-1} \Pi \\ &= \Pi X^{-T} C^\dagger X^{-1} \Pi. \end{aligned}$$

To verify (43), it suffices to verify that $B\Pi X^{-T}C^\dagger X^{-1}\Pi$ is symmetric, which we now do:

$$\begin{aligned}
B(\Pi X^{-T}C^\dagger X^{-1}\Pi) &= \Pi B X^{-T}C^\dagger X^{-1}\Pi \\
&= \Pi(XCX^T)(X^{-T}C^\dagger X^{-1}\Pi) \\
&= \Pi XCC^\dagger X^{-1}\Pi \\
&= B^\dagger BXCC^\dagger X^{-1}BB^\dagger \\
&= B^\dagger XCX^T XCC^\dagger X^{-1}XCX^T B^\dagger \\
&= B^\dagger XCX^T XCC^\dagger CX^T B^\dagger \\
&= B^\dagger XCX^T XCX^T B^\dagger \\
&= B^\dagger BBB^\dagger,
\end{aligned}$$

which is symmetric as B is symmetric.

As B and $\Pi X^{-T}C^\dagger X^{-1}\Pi$ are symmetric, it follows that (44) is satisfied as well.